Evaluating Understanding in Cross-modal Multi-encoder Models

Stephen Daniel Mander, BSc (Hons)
School of Computing and Communications
Lancaster University

A thesis submitted for the degree of
Doctor of Philosophy

April, 2025

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis does not exceed the maximum permitted word length of 80,000 words, including appendices and footnotes, but excluding the bibliography.
Stephen (Daniel) Mander

Evaluating Understanding in Cross-modal Multi-encoder Models

Stephen (Daniel) Mander, BSc (Hons).
School of Computing and Communications, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy*. April, 2025

Abstract

Ensuring that AI benefits all, not just the anglo-phonic, nor the big companies, and remains sustainable covers a multitude of different problems in computing. Low-resource language is an area of NLP studying the difficulty of training models with minimal data, annotation or resource (which can include hardware/ financial motivation). Limits to data quantity challenge many training approaches in machine learning, reducing models' ability to generalise and understand text.

This thesis discusses how using multimodal models (those that can use images AND text to learn) can provide a much-needed grounding to the issue of understanding text. The introduction will approach this through the lens of detecting hate speech, a problem that typically disadvantages low-resource domains. After establishing the merits of using autoencoder approaches, the main body of work will focus on a training paradigm to bring the large-scale approaches to be replicable. In the subsequent chapters, alternative perspectives are considered, looking at the viability of the assignment assumptions that are being made to further boost gradients.

This thesis addresses some of these problems by exploring some of the deeper rules underlying these generative methods. Learning low-resource languages is explored in this work as a challenge of scale: A paradigm is presented for training in an ecologically feasible and academically affordable manner. By reengineering previous methods, scale is addressed and a new algorithm for efficient training is presented. By altering the calculations and demonstrating how to add an additional encoder, this work provides a stepping stone to a greener, academically viable, and open model for low-resource domains. By addressing novel approaches to training low-resource language, this thesis also explores a different way to view training as an assignment problem, which can be abstracted, approximated and provide efficiency improvements to the more general training frameworks for both NLP and computer vision.

Publications

The following publications are derived from my PhD research:

Stephen Mander, Scott Piao, and Hossein Rahmani. "Contrastive Training with more data". In: *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023*. Ed. by Krystal Maughan, Rosanne Liu, and Thomas F. Burns. OpenReview.net, 2023. URL: https://openreview.net/forum?id=ZTp85mW5nFy

Stephen Mander and Jesse Phillips. "LiSAScore: Exploring Linear Sum Assignment on BertScore". In: *Natural Language Processing and Information Systems*. Ed. by Amon Rapp, Luigi Di Caro, Farid Meziane, and Vijayan Sugumaran. Cham: Springer Nature Switzerland, 2024, pp. 249–257. ISBN: 978-3-031-70242-6

Contents

List of Figures

List of Tables

Chapter 1

Introduction

'Machine Learning' and 'AI' have rapidly become buzz-terms in recent years, signifying a system that uses complex mathematical approximations of problems to intelligently learn a problem space, with complex rules that cannot be readily coded or written. Natural Language Processing (NLP) and Computer Vision (CV) are 2 major areas within machine learning. Many of the popular methods underlying these areas need significant quantities of data to effectively model specific tasks or domains as complicated as language or vision. As these terms captivate the public imagination, it is important to verify the breakthroughs and gain a better awareness of the capabilities these systems exhibit, which is summed up in this work as 'understanding'.

This thesis explores how well computationally generated embeddings truly understand what they represent. The thesis demonstrates the performance of multiple encoders to improve the economies of training available in low-resource contexts, in the data and hardware domains. Subsequent chapters outline the performance of different methods in higher-dimensional contrastive training, as well as offer a cursory exploration of the myriad applications of this work.

To begin this body of work, it is worth discussing the fundamentals of why certain tasks remain difficult in the machine learning space, even with increasingly complicated LLMs available.

The journey behind this thesis began with the task of hate speech detection. Making the internet a welcoming, inclusive space involves trawling through trillions of words, and removing anything that may cause offense. The noble goal of being able to isolate negative speech is quickly hindered by the myriad complexities of processing such quantities of data, identifying or defining hate speech, and being able to remove it. The detection task simply tackles the problem of identifying it. Even for such a simple task, a definition for hate speech must be agreed upon. The definition must take into account many different factors: Language (not just English and how this varies by person and context); Context within the flow of conversation;

and subject area that the words are spoken to. There are many comical examples of filters naively removing words from the lexicon of users to the detriment of innocent conversations. (Removing the word 'bone' as a euphemism can be deeply detrimental to a paleontology conference)

In this work, the challenge of what words mean is isolated and considered as a deeper problem; it is not so simple as just a classification or word embedding problem. Dictionary curation is slow, requiring much human effort and scales poorly to web-scale data and is insensitive to context. A system is needed that can spot the changing meanings and contexts of words.

Equal critique can be applied to how computers use language in a generative context. Some sequences of generated text may seem intrinsically insincere. In such a case, it highlights why semantic and epistemological understanding of language is so difficult; the meaning to be conveyed often betrays the meaning of individual words. An example would be a generated string of text confirming the flight of the first pigs. A reader may spot the semantic parallel to historical metaphors and hyperbole around flying pigs and would naturally assume insincerity. More prosaically, an AI writing about whether a deity exists (and in so doing selecting a belief system), is naturally assumed to be data bias or nonsense – yet enforcing that a model shouldn't have a belief system seems laughable if there's no chance of it occurring naturally. In the semantic parallels of language appearing questionable upon application, illustrate the domain of semantics, or the examination of word meanings.

Investigating the study of what words mean is an indicative factor in understanding why there is so little annotated information for detecting hate-speech. It seems a simple task: processing a natural language input and making a determination. In actuality, hate (and its detection) is a subjective concept - it relies on an individual taking offense. If a system is only observing what is written, one cannot know that no other reader will perceive offense, nor necessarily whether offense was meant in the first place. This perhaps belies why there are relatively few datasets claiming gold standard annotation for this task.

To demonstrate the focus of this thesis, a statement like 'you're a banana' is worth considering. It is a phrase that many in a western audience may have even heard growing up, indicating the silliness or absurdity of one's actions. However, to other audiences, this phrase may be dripping with malice, the inflammatory accusation being that someone is 'white on the inside'.

That implicit understanding of why something may be inflammatory in some contexts and not others is a complicated reasoning task, which several approaches attempt in supervised ways, using annotated associations and connections [93]; however, the understanding and connection that something may be offensive is a problem that extends beyond the domain of Natural Language Processing (NLP).

The thousands of properties that every object has make the combinatorics

and reasoning implicit to metaphors incredibly difficult to annotate for in labelled examples. Recent philosophers such as Heidegger to as far back as Plato have understood objects as having near-infinite properties.

This work therefore will largely neglect supervised approaches and explore unsupervised learning to attempt to learn the myriad properties that surpass annotation. Unsupervised learning is a paradigm where a model does not learn from human labels; instead, it learns the salient features of the input to determine how to represent the concept in latent space. Conceptually, the idea of a distribution of embeddings across high dimensions allows a similarly infinite number of values a word can be encoded into. Thus, it allows us to approximate and query the infinite permutations and properties of objects. However, just because something can be queried, does not guarantee the model used is encapsulating all the potential properties, uses, or abilities of objects. This thesis shall explore whether all the encapsulation of properties can be measured.

1.1 Defining of Terms

There will be many terms used in this body of work that span disciplines or might be used in natural language with an imprecise computational meaning. For the purposes of this document, the following simple definitions will be adhered to:

- **Semantics**: The study of what a token represents or means, often used in relation to NLP discussing the meaning of individual words, or in CV as to the prototypical class of an object.
- **Semiotics**: The study of meaning, which transcends individual tokens to convey concepts or meaning beyond the sum of tokens.
- **Knowledge**: A stored statement of justifiable truth.
- **Human / humanistic**: A reference to the typical human experience, both in having normative 'average' intelligence and 'average' potential to learn.
- **Understanding**: The ability to create robust representations of a concept that capture the important features as pertains to context which behaves conceptually as the base concept.
- **Reasoning**: The ability to apply logic to both the observed, and the respresentations of, concepts with the goal of a predicted outcome or behaviour.
- **Model**: An approximation of a system. In Computing models typically require training to adjust their behaviour and properties towards an optimal behaviour, often relying on hidden representations and to process an input.

1.2 Research Goals

The motivation for this research stems from the increasing complexity and opacity of modern machine learning models, particularly those involving encoders in vision-language tasks. While existing encoder models have demonstrated impressive performance, there is a critical gap in their ability to form truly holistic and interpretable embeddings that align with human understanding. This gap raises a pivotal question within limited resource contexts: to what extent can we ensure that these models comprehend the underlying semantic structure of the data they process? Consequently, this thesis aims to investigate how we can better understand, evaluate, and improve these models to achieve more robust and interpretable embeddings. Specifically, by addressing the challenge of defining and measuring semantic awareness in model embeddings (**RO.1**), as well as evaluating models in the absence of clear ground truths (**RO.2**). Additionally, we examine the practical limits of scaling these models within constrained computational resources, which is crucial for ensuring academic replicability (**RO.3**). Each of these objectives directly ties to a central question of how to effectively create, measure, and scale models that not only perform well but also offer meaningful, interpretable outputs. This inquiry is foundational to answering several critical research questions, including: how we can isolate specific elements of knowledge within machine learning models (**RQ.1**), how to facilitate model training and replication within limited computational resources (**RQ.2**), and how annotation assignment can improve model performance on both micro and macro scales (**RQ.3**). Furthermore, we will explore the impacts of noisy approximations on grounding and assignment within models, as these issues are crucial to understanding the limits of a model's comprehension and its ability to exhibit true semantic awareness (**RQ.4**). Together, these objectives and questions form a cohesive framework for evaluating the effectiveness of encoder models and advancing their interpretability in real-world applications.

1.3 Research Objectives and Questions

RO.1 Present a training paradigm that is not tied to a domain prediction but can create meaningful embeddings. The embeddings generated in the vision-language space should have a depth of understanding that users can trust is fully aware and comprehending of concepts. The goal is to create and define a methodology for a semantic and semiotic awareness to an embedding space. Given the tendency of models to be re-used on downstream applications, it is imperative our tests show a robust, dataset-independent method for verifying training and learning of concepts.

RO.2 Show evaluation methods available where there is no clear holistic ground truth for what the model should have learned. Models in the encoding-only space offer

novelty in training: they consume data without predicting an output. Challenges are presented in evaluating such a model without inferring assumptions such as a downstream distribution. Different evaluation metrics are available and will be deployed during training, but this research will show simple methods afforded to some training paradigms that do not require separate, distinct evaluation workflows. By deploying several metrics, the performance of metrics can be identified partially by consensus.

RO.3 Demonstrate the limits of scaling within the limits of 24GPUHours, and the projected performance increase for future work. Core to the remit of this work is the confines to university scaling. At its conception, the maximum available allocation was 12GPUHours. During that limit, the SOTA approach in this domain was used on 512GPUs for 19 days. This poses significant academic concerns over the lack of reproducibility and replicability. This work therefore seeks to understand the scaling rules present by batch size and how that is felt at different training scales.

The following research questions (**RQ**s) which will be answered in each chapter:

RQ.1 How can elements of knowledge be isolated within a machine learning model? This will be answered in Chapter 3, looking at how low-resource language tasks can be effectively taught.

RQ.2 Many state-of-the-art models exist beyond the training capabilities of academic replication. How can model training be reinvented to facilitate replication at a smaller scale of compute AND data? Chapters 5 will answer where the limits are for such models and, moreover, how we can prove the effectiveness at different scales, focusing on replication within the limits of 24 GPU-Hours.

RQ.3 There are many advantages available to models that use weak and pairwise supervision to learn the relevant associations. Can annotation assignment have improvements on a micro scale with performance gains comparable to weak supervision on a macro scale? Chapter 6 will look at other methods that use similar assignment-based metrics and will use the supporting publication track to isolate performance benefits associated with modifying the assignment criteria.

RQ.4 Many of the semiotic studies that underpin **RO.1** point to grounding and assignment as fundamentals in attributing to a model 'understanding'. What are the impacts of approximating these assignments or using noisy approximations?

1.4 Thesis Structure

This document will begin with a deeper discussion of some of the background and related methods to this body of work. Latter chapters will present an AI approach

that can surpass the limitations of supervised and granular annotations and exhibit capabilities that demonstrate higher-order knowledge.

In Chapter 2, the background and context of this work are discussed and explored. The problem space is addressed, and some benchmarks are presented. The first chapter seeks to demonstrate the way these models work, presenting an intuitive guide to why semantic embeddings do not have the same mathematical relations as simpler embedding spaces that may be more familiar to readers.

Chapter 3 begins to explore the applications for low-resource languages. Experiments are outlined for how these multi-modal models can improve training for low-resource languages by providing alignment.

Chapter 4 defines the challenges of evaluating training. Some of the model components presented in this research may present a novelty in not returning a natural embedding in a form that is readable. This chapter, therefore, discusses which metrics can be used and how they combine to guide training paradigms.

The penultimate chapter details the limits of scaling contrastive training, extending the training methods presented in Chapter 3. Novel calculations are presented to address some of the mathematical curiosities, and the labels for these algorithms are explored.

In the final section, Chapter 6, another way of improving low-resource training is explored: treating training as an assignment problem. Existing methods are improved with application-specific optimisations, and the performance profiles of inputs are explored, which present opportunities for optimisation in other contexts.

Chapter 2

Background and Related work

2.1 Background

This thesis explores how well computationally generated embeddings truly understand what they represent. But an introduction to standard techniques is imperative to explain the state-of-the-art models built upon. Machine learning (ML) is a well-established technique for predicting language output from a wide variety of modalities. It is now well established that the context of an image provides valuable information to inform and improve object predictions. The thesis demonstrates the performance of multiple encoders to improve the economies of training available in low-resource contexts, in the data and hardware domains. Subsequent chapters outline the performance of different methods in higher-dimensional contrastive training, as well as offer a cursory exploration of the myriad applications of this work.

2.1.1 Word2Vec and Transformers

One of the early approaches to understanding language was the idea that a word can be defined by the company it keeps. The approach was to split a sentence up into tuples, or (a set of tokens) often called n-grams where 'n' is an integer referring to the spacing of the word. These pairs are then encoded as a set of one-hot vectors, the index of '1' in the location corresponding to the word. By projecting these vectors into an embedding space, the size of each vector is reduced from the size of the vocabulary, down to a more computationally efficient number (like 512 or 128). This causes words that have similar meanings, (and likely a similar context) to tend to cluster together.

Subsequently, models have increased in size and complexity, culminating in LLMs, which power the majority of popular models. LLMs are predicated on a transformer architecture, popularised by BERT [120], but exist at a significantly larger scale, and now use more complex training techniques including mix-of-expert approaches

and reinforcement learning. However, the core technological advancement of the architecture is the use of attention [116], allowing tokens in a sequence to bi-directionally affect the meaning of other tokens. Crucially, the representations of each token are modified by the others.

2.1.2 How Computer Vision Understands

The wider task of this thesis is to explore the way that model representations understand, or rather, generate holistically queriable representations. Traditionally, the task of computer vision is to receive an input image and classify it. Many examples might use a simple set of classes like 'Cat' and 'Dog', and typically, every image is exclusively a cat or dog. This task can readily be expanded to having an 'Other' category to perform as a detector, or have some architectural changes to be able to draw a bounding box around the cats or dogs.

The challenge for this body of work is to take a simple example like this and question whether it really understands the concepts beneath the classification 'cat' or 'dog'? Most classification tasks do not concern themselves with explaining the learned concept, but rather present a confidence score of whether the image fits within the training set of cats, or belongs closer to the training set of dogs.

Because no concepts are codified, the problem of classification scales poorly with the number of classes. This is partially because there are multiple clusters in an embedding space for points to move between, and the balance of data must be accounted for. However, a cornerstone paper [37] used Deep Visual Semantic Embeddings (DeViSE) to scale effortlessly from 1k to 21k classes. Scaling to so many classes with few-shot learning was achieved by assuming that the classes would relate to each other in the same way that the labels for each class do. The model used the geometry of GloVe [101] embeddings (the previously discussed W2V model trained on web-scale data).

This was arguably one of the first effective attempts at ensuring a computer vision framework had some knowledge of what was being classified or predicted: the class name affects classification.

DeViSE paved the way for a plethora of other models, with architectural similarities seen in more modern DETR [138, 61] and DETIC [137] models that use transformers for class comprehension and computer vision.

This work will continue the approach of using language embeddings as a basis for building holistic representations. The test, however, will have to be in a zero-shot context, as this eliminates the ability of a model to be reliant on the training set.

2.1.3 How to Define Representational Robustness

The goal of this research is to show how cross-modal tasks (using linguistic and visual domains) are the key to better representations that can be said to 'understand'. The use of cross-modal training is to remove the abstraction that all learning is a classification task. Fundamentally, to know a model is building a correct, robust representation, the model must be queriable against an open domain. (I.E. not task limited)

Prior work includes GloVE, a generation of 'universal' embeddings that are said to carry this property. It relies on the assumption that web-scale datasets are a good source of definitions. Comparatively, DeViSE assumes definitions that are implicit in embedding space geometry, have been tried and tested on enough data they must be correct and precise. However, there is little literature on why such wide samples of data can simply be aggregated during training. The definition of 'web-scale' now is significantly larger and changes more dynamically than over a decade ago when those methods were proposed. The rapidly changing and evolving space heralds new challenges in NLP around words that may not have warranted dictionary definitions, or have niche, specialist, or contextual meanings. W2V approaches are inaccurate in these cases where contexts have unique languages or dialects. In such cases, building a representation from such infrequent tokens is problematic.

The challenge of robustness to context also applies to computer vision, where traditional class-based approaches can struggle to be queried for anything other than labels. There are very few open-set (where there is no limit to the number of classes) challenges, and using small datasets to represent a robust space, such as can be aligned to a vector space as in the GloVE or DeViSE paper, has only been realized in work contemporary with this thesis, in the form of DETIC, which relies on downstream training from the work presented.

An alternative perspective on this domain is provided in the study of semiotics: the study of signs and symbols and their use or interpretation. Semiotics suggests that tokens often refer to an object by a set of properties or multiple names, which allows us to communicate meaning. Referring to a table as 'a flat surface' or 'desk' conveys the desire to work - either by the property that facilitates work, or by the primary use of the object. This is a challenge to computer vision that relies on clustering because the set of objects that receive the classification 'desk' may be different but overlap with 'table' and 'flat surface'. The set of properties that a particular class has is usually referred to as a homeostatic property cluster: No one property defines the class, but the presence of enough of them is indicative of the vocabulary used to describe an object.

In the classic example of a car missing wheels, a car is still a car if it is missing a component, even when these are a critical part of the functionality. This example is offered to emphasise that the computer vision approach to understanding this scene is

complex: it predicates on having a sufficient understanding of an object to understand the purpose or potential of an object even if critical components are missing. Holistic representations are therefore critical to contexts, where a car missing wheels might be a critical detail depending on where it is observed.

Representations of an image, and the contents, need to be sufficient that for open-set works, objects can have myriad descriptors that can match any visible object. In other words, that objects in the space can be queried or predictably found based on sets of properties [71]. Typically this can be considered as the embedding space preserving orders of knowledge. This can be demonstrated in the classic word2vec example that 'Man' is to 'Woman' as 'King' is to 'Queen'. In the example, the relation between 2 points can be considered a single order of knowledge. That it pertains to another pair is second-order knowledge, realizing that this relates to other like properties is a third order and so on.

A way of enforcing orders of knowledge, modern transformer architectures use Attention : that tokens can be projected as a Query, Key, Value tuple, and used to create a representation of a value according to Query and Key. This view of a representation and stepped attention over it is akin to building a tool to consider where it belongs in an embedding space and viewing a point as a function of the clusters it sits in. The only caveat is that where philosophically every object has a near infinite set of properties, models constrain the dimensions of hidden representations to finite numbers (usually integer powers of 2).

2.1.4 Understanding in Low-Resource Languages

The preceding section discussed the use of representations as an encoding of information about a visual concept. In a computer vision context, building representations as capable of conveying all queryable information in a scene is critical. To avoid the trappings of a large number of classes, an open domain must be used. However, given limitations to the scale of this research, understanding must be viewed through the lens of low-resource language. Languages are considered low-resource whenever a language is lacking large monolingual or parallel corpora and/or manually crafted linguistic resources sufficient for building statistical NLP applications; it is considered a low-resource language. In a computer science context, this definition extends to including languages with only a limited number of gold-standard annotations or the (un)availability of wider corpus. From such small samples, gaining a good understanding of the language is problematic without prior knowledge.

LLMs exhibit excellent ability to learn low-resource languages with prompting and examples [13], due to wider world knowledge. Therefore, what may be deemed as 'understanding' goes beyond the standard data extraction used in old NLP; it no longer suffices to apply data science to POS tags or extract key words because without

correct interpretation and white-box systems, they are meaningless. Even with the grammar, structure, and objective definition of words, the nuance of language may still be lost. Modern approaches with such methods often resolve meaning through LLMs to associate generated labels with more fitting and interpretable labels. As the user skill/interpretability threshold is reduced, the risk of poor design and poor usability in our models increases. There is an ever-growing laundry list of models that fail to meet real-world standards: Idioms, metaphors, and hyperbole regularly escape researchers' ambitions of 'solving' language. This is part of the reason that open-set language remains difficult - for the affirmative case, there are 100s of different methods of expressing it, and using word-level and token-level loss is no longer adequate when sentence-level meanings are needed. This also rejects the issues that confront low-resource languages, where not only is the amount of data available so low that models are difficult to train, but the prevalence of LLMs that provide shaky translation pollutes the space.

In recent years, UNITER [20], CLIP [103] and many other cross-modal approaches [79] show that open-set language is solved - especially as these have become the SOTA backbone for other methods like DETIC [138], Flamingo [4], DALLE [73] and many more. It is worth a consideration of scale - these methods require training measured in GPU/TPU years. The concept of some languages being solved is a question of scale, application, and data: the 3 things that this work lacked at conception.

The breakthroughs discussed in computer vision previously that stem from aligning classification with language are almost entirely anglo-phonic. There are very few datasets or works that exist in other languages that use richly annotated datasets in other languages. This is why this work seeks to explore training paradigms that can leverage vision-language grounding to improve the representations curated in low-resource contexts.

2.1.5 How CLIP Works

In the last 2 years, CLIP has radically altered the Machine Learning space. Each component of the CLIP model is nothing new, but the combination of 2 encoders projected to a shared space is an entirely novel phenomenon. The shared encoding space, enabling the presence of both visual and text modalities to be projected together, has been a crucial building block for tools like Dalle, StableDiffusion, Dalle2 and many more generative models.

The CLIP algorithm works by contrastively training samples in an effectively unsupervised manner. During training, a batch of items consists of image and text components I and T, respectively. These are then encoded separately, and loss is calculated by comparing each encoder's output with cosine similarity and maximizing the similarity for corresponding items with cross-Entropy Loss.

Pairing items in batches may seem obfuscated. So here is the flow of data: Imagine 2 auto-encoders that, given a batched input, create an output of uniform random features.

When plotted on a graph, these features appear as follows.

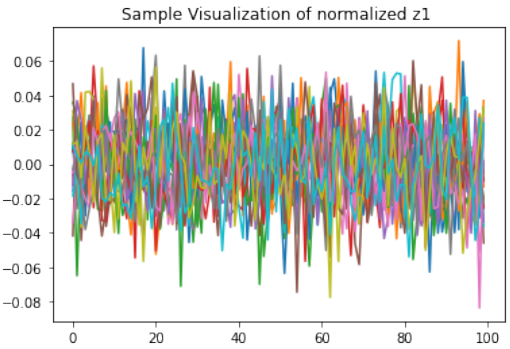


Figure 2.1: This plot shows what a single batch of normalised vectors in an example input may look like when overlayed

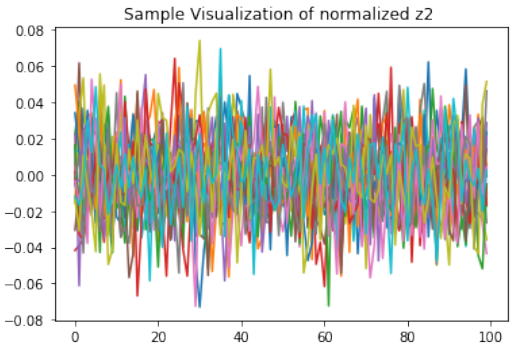


Figure 2.2: This plot shows what a second batch of normalised vectors in an example input may look like when overlayed, notably different from Figure 2.1

These may be difficult to distinguish to the human eye, but when compared by multiplying, they produce a set of logits (a batched form of cosine similarity). for comparison, when we produce the same graph using identical inputs, the result is as follows: shown in Figure 2.4.

The distinct spikes indicate the cosine similarity of every item in the first batch, corresponding identically to that in the second batch.

Notably, the minor deviations in peak heights are not smoothed by averaging with the transpose of the logits but rather are a reflection that the inputs are non-linear. In the case of dual encoders, it follows that items in the first encoder are independent of items in the second encoder and vice versa. This symmetry is largely only found in identical inputs; in real-world data, it is unlikely. This can be stated as: $P(A_i|B_j) \neq P(B_i|A_j)$ unless $A_i = B_i$ and $A_j = B_j$)

One novelty of this training methodology is the scaling: contrastive training requires a batch size greater than 2 to function. Furthermore, the greater the batch size, the greater the number of in-batch negatives. This has been largely criticised because it solidly places AI research and validation in the hands of large companies or organisations with huge training resources. The original model took some 33GPU years to train. Aside from the ecological impact, this is a prohibitive time scale unless performed at enterprise scales.

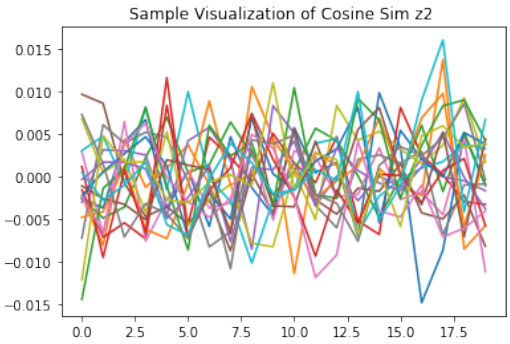


Figure 2.3: The comparative similarity of each item in a batch to every other item (note the x-axis change in scale).

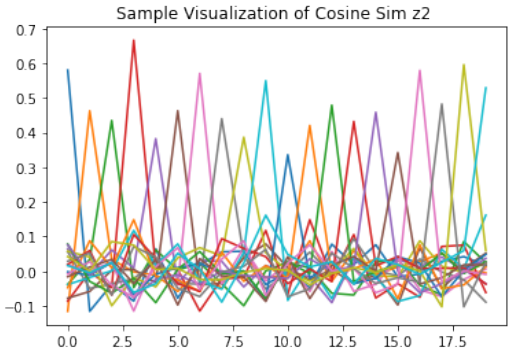


Figure 2.4: A plot showing the comparative similarity when the 2 batches match each other, each line has its own distinctive peak which corresponds to order within the batch.

The subsequent work aims to reduce that. Tying into the introduction, considering understanding and embedding spaces in high dimensions, instead of confining individual points by 2 projections, allowing them to move accordingly, this work explores using greater numbers of dimensions.

The goal here is to introduce additional data to add additional dimensional constraints. We consider what happens when a batch consists of $I, T_0, T_..., T_n$. Naturally, such a set might undergo different encoder architectures depending on the modality. However, for the sake of proof of concept, we train the $'theta'_T$ parameters by guiding the representations of all inputs $T_0, ...T_n$. However, in contrast to training these samples, we create an n-dimensional loss table. This shows far better scaling for training for a given batch size.

```
function make_Logits(I, C1):
# Normalize image and text features
I = I / norm(I, dimension=-1, keep_dimension=True)
C1 = C1 / norm(C1, dimension=-1, keep_dimension=True)

# Calculate logits
logits_per_image = dot_product(I, transpose(C1))
logits_per_text = dot_product(C1, transpose(I))

# Calculate loss and return the result
return logits_per_image * exponential(logit_scale), ↩
   logits_per_text * exponential(logit_scale)
```

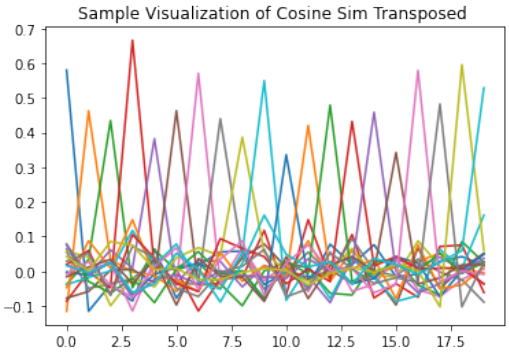


Figure 2.5: A graph showing the similarity with each batch transposed, bearing identical shape to Figure 2.4, thus proving the matrix multiplication of these batches will preserve a gradient in both directions

A good way to understand the effects of contrastive training in high-dimensional vectors is to ascertain the significance level of 2 vectors of the same magnitude having a good Cosine Similarity.

In 2 dimensions, we have the simultaneous equations:

$$v_1 = (x_1, y_1), v_2 = (x_2, y_2)$$

$$\sqrt{x_1^2 + y_1^2} = 1$$

$$\sqrt{x_2^2 + y_2^2} = 1$$

$$x_1.x_2 + y_1.y_2 > 0.5$$

The value of 0.5 has been picked here because it is easily recognizable geometrically.

To simplify the problem and plot the values on a graph that satisfy this equation, choose a simple value of (v_2) that simplifies the above equation: where

$$v_2 = (0, 1)$$

$$x_1.0 + y_1.1 > 0.5$$

simplifies to $y_1 > 0.5$ These means can be mathematically calculated with the identity of $v_1 = (x_1, y_1)$ where $\sqrt{x_1^2 + y_1^2} = 1$. Or, on a graph, we can say that the angle at which the threshold is at v_2 is θ where $arccos(0.5) = \theta$

When we use a threshold of 0.5, as the number of dimensions of our vector,n, increases, we find that the significance level for a cosine similarity ≥ 0.5 becomes 0.5 follows $1/n + 1$.

θ	Cosine Similarity	$n = 2$	$n = 3$	$n = 4$	$n = 5$
π	0	0.5	0.5	0.5	0.5
1.47063	0.1	0.46870	0.45000	0.43607	0.42097
1.36944	0.2	0.43609	0.39992	0.37329	0.35009
1.26610	0.3	0.40323	0.35006	0.31185	0.28078
1.15928	0.4	0.36906	0.30002	0.25228	0.21549
1.04720	0.5	0.33333	0.25003	0.19551	0.15554
0.92730	0.6	0.29499	0.19996	0.14243	0.10421
0.79540	0.7	0.25456	0.15024	0.09414	0.06091
0.64350	0.8	0.20500	0.10007	0.05210	0.02820
0.45103	0.9	0.14397	0.05018	0.01873	0.00739

Table 2.1: For an n-dimensional sphere, this table shows θ, an angle at the centre, compared to the volume, (or equally surface area), contained within the angle. The experimental approximations of these volume of n-dimensional sphere for different threshold values of cosine similarity rounded to 5.S.F. This shows the decrease in probability as n increases, showing why this method is especially effective in high-dimensions

2.2 Related Works

As will hopefully be well known to any researcher in this or adjacent fields, 2020-present has been a turbulent and exciting time seeing a rise from machine learning to advanced deep-learning systems. Beneath the confusion and controversy, AI has been obfuscated by the myriad systems claiming to be AI; there are well-coded databases, distributed models each with radically different domains, and the others all claiming the illusive goal of intelligence. Against this backdrop, cataloguing the contemporary works that claim novelty, even down to those being based off the same model,CLIP [105], is a near impossible feat. It has been cited over 26,000 times and is considered foundational to many differing domains and applications.

There are many works that seek to improve, use and replicate CLIP, even offering enhanced refinement of the training pipeline by reintroducing supervision via region selection with an RPN component [136]. Approaches like this emphasize the computer vision application over the holistic language capability. There are other works [24], that emphasize similar capabilities in less supervised fashions, representing semantic masks with embeddings. However, such works and others gently neglect how hierarchical object detection works.

One key advancement in the field, published 2 years after the main body of research, is Llip [70], which uses the same training paradigm of multiple captions per image to significantly improve the zero-shot capability. They focus on combining

captions with visual tokens with cross attention across each caption. Then they used the output through a more traditional clip-style contrastive loss. While the results are impressive and show potential, it should still be noted that CommonCrawl [15] datasets were used, which are far larger than this work can manage: 2.5 Billion image-caption pairs, pre-trained on a larger 12.8B pairs. Compared to the original 512 GPUs, they only used 128 and 256 larger A100s for a bigger batch size in their experiments. That work not only validates the premise of this thesis, but also proves that it scales unquestionably to bigger training pipelines.

On a smaller scale, the works focus on very fine-grained detection rather than semantic comprehension in open-vocabulary contexts [126]. This builds on prior work using a 2-stage detector aligned with a multi-modal transformer that receives both image and text tokens. However, this test is still constrained to the classification of nouns rather than the true open set testing. It does not demonstrate semantic capability or nuance in a way that might theoretically be present.

In the last year, SigLIP has been proposed [128], suggesting the use of sigmoid loss over cross entropy for loss. The key advantages of this approach are rapidity in scaling and batching, allowing scaling across multiple devices asynchronously. The work validates several key contributions of this work: that the exact activation function is not strictly relevant and that scaling rules are a key factor in model training and overall efficacy. Although the sigmoid loss performs equally well and faster than stock CLIP (though not compared on the same hardware), by removing the reliance on contrastive training on logits, it may affect training. It is unclear due to other training differences whether this removal of the graph is significant in its impact on training and downstream applications. Specifically, it can be argued that because the gradient for a given logit is not only being minimised or maximised, it is not clear in which direction it will move: contrastive training would ensure it moves towards a single point and away from B other points. The counterargument is that when B is large enough, this is not a relevant concern.

An alternative approach to increasing batch size despite hardware limitations is to investigate the use of pre-computed values that are detached from their gradients [39]. Normally, calculate m mini-batches per batch and aggregate the m^2 gradients to approximate the batch between devices. This is similar to the abstraction that occurs in transformer heads to accelerate transformer operations. Precalculating features and, therefore, gradients requires a significant RAM footprint and execution time overhead. The work is notable for being at a scale similar to this work, but lacking in training details or scale, and therefore, even though there is a marginal speed overhead, it is unclear whether the synthetic batch sizes achieved are meaningful.

Many other works have attempted to replicate CLIP at various scales [80], including attempts to alter how multiple GPUs are used in training, or using simpler approximations of loss. Such attempts are contemporary with this work.

2.2.1 Related works in Low resource domains

In recent years, LLMs (GPT-x [82], BERT [30], ELECTRA [22], LLama [115], to name a few) have boasted phenomenal zero-shot and few-shot performance in low-resource domains. Even approaches that rely on only altering a few layers with finetuning have shown exceptional results [59]. However, the difficulty for the extreme cases of low-resource languages is that many underserved languages share minimal philological heritage with others - meaning the structure, syntax and form of the languages are fundamentally different. Consequently, structures, grammars, phonemes, and characters are used in significantly different manners and represent barriers to models being adapted readily across languages. It is widely held that the intermediate layers of a network enable the performance of reasoning and meaning extraction from the text, processing the input domain to an output domain, whether that be instruction following or masked language modeling. These intermediate layers, with attention mechanisms sculpted by positional embeddings pertaining to a specific language, have no reason to be transferable across contexts.

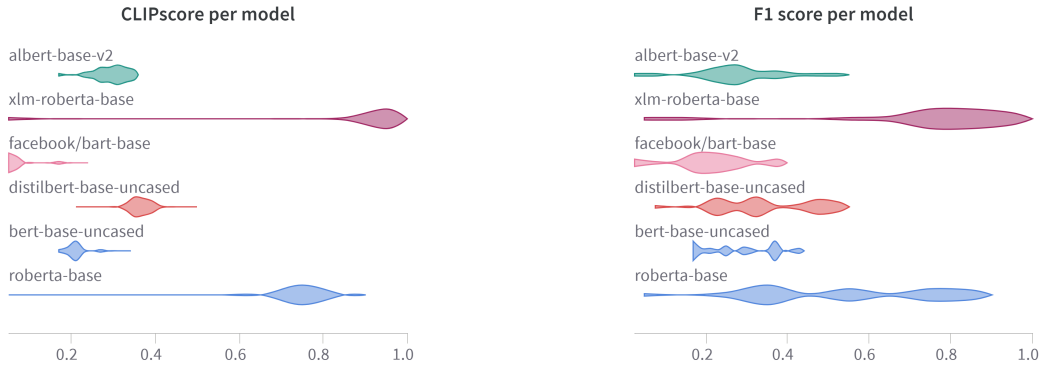
There are epistemological issues with assuming that many of these languages share any common structures or ideas with those that fit well with LLMs. Many of the problem areas for detecting hate speech or metaphor-infused language can be derived from a lack of visual grounding. Idioms, emotive and visual language are largely cultural and vary massively across cultures, languages, and even dialects. It is the goal of this chapter to attempt a visual grounding approach to teaching low-resource language, to maintain this grounding. This can be shown by the work translating CLIP into other languages [14], that the visual domain does not have the same parallels to a purely linguistic one, but rather the geometric constraints are not all subjective.

The underlying assumption that languages occupy a subset of universal language must be challenged. If this were true, it would be fair to assume that there was a set of rules that govern the geometric transformation of one text to another. Instead, the use of Computer Vision as a grounding domain will improve translation through the use of weak-noisy labels. The concept of visual grounding means that the captions do not have to agree but are likely to have semantic similarities, which is sufficient to train from. This approach, akin to many NMT (Neural Machine Translation) ideas [112], abstracts away from the traditional word-to-word or sentence-piece approaches that dominate the literature. A BERTScore [132],[54] like approach is needed: to aim for a semiotic match, not token conversion.

2.2.2 Related Analysis Methods

Approaches like BERTScore work on the assumption that intermediate representations point to single, comparable tokens. The approach underlying BERTScore [132,

54] assumes that there is a permuted one-to-one mapping of tokens with comparable semantics. As discussed earlier, the attribution of language as being a sum of its parts is to ignore the semiotics and meaning inferred. Therefore, analysis of translation or language tasks ought not to rely on equivalency between sets of tokens. Such methods can be further discounted because of the clip architecture that uses [EOS] (End Of Sequence) token as a semantic summarization, referred to as CLIPScore. However, in testing for LiSAScore, a work culminating from this chapter and Chapter 6, this was shown to be a good approximation to BERTScore, as shown in Figure 2.6, showing the correlation between them, using a dataset that only some of the trialed models were trained on, to profile the correlation of the metrics across a range of performance.



(a) Model scores over the WMT dataset measures with BERTScore

(b) Model scores over the WMT dataset measures with CLIPScore

Figure 2.6: A figure showing that CLIPScore and BERTScore produce very similar results across language models on the WMT dataset

Figure 2.6 shows how there is a strong correlation between the 2 metrics, where one is a significantly smaller calculation. In this chapter, other evaluation metrics will be explored for evaluating how training can be measured in low-resource contexts.

Prior training to visual grounding of the embeddings could theoretically yield false positives: captions that are semantically apart but might describe the same image. This explains the use of cosine similarity as a non-directional proximity metric.

Works like BERTScore can be applied to intermediate layers too. However, these modes of operation have also been discounted due to computational complexity and disparity between encoder architectures between modalities. Such works show the potency of intermediate layers for accurate comparison. In this chapter, optimizations are presented to metrics that can apply to heterogeneous architectures.

2.2.3 Related Works for n-dimensional Training

In recent years, much work has been done on distilling LMs for efficiency and to reduce the overhead associated with fine-tuning large models [17][5][48]. It is clear from the body of existing research that grounding language in a visual domain is a very important component of understanding and is key to capabilities like concept referencing and visual and spatial reasoning. In the low-resource language domain, there is a wide body of work looking to use pre-trained models to rapidly demonstrate viable translation models. NMT-based approaches [112] seek to build embedding spaces from context embeddings, stating that they have a similar topology to which a small set of samples can anchor to the target domain. There are other approaches based on fine-tuning that attempt to alter the projection of a space onto another. This has been frequently and successfully applied to CLIP-based systems as a way of very quickly tailoring models to a new language, even by just fine-tuning a single projection or encoder [14].

There are also works that replicate these findings in spoken linguistic sources, although these are not direct parallels for the implications due to the addition of voicing, tonality, and intonation. The significance of Dalmia et al. [27]'s LegoNN is to demonstrate not only an interchangeable intermediate embedding space, but one that models can be trained to directly. This poses significant ramifications for our approach, seeking to replicate a similar result at smaller scales in an encoder-only model.

If the goal here was to emphasise the importance of simply finding an intermediate embedding space, the emphasis would be training translation models using a BERTScore-style metric. However, there are many practical limitations, such as the additional overhead of a whole extra model on a system that presents millions of parameters to track gradients through. If the code were sufficiently adapted to add a contrastive loss, which would favour sequences that had varying semantics of tokens, which fits the majority of cases.

The limitations of these works are that they do not appreciate the nuances of language that impact grammar, emphasis, or cultural norms that pull out different features, especially in smaller sequences. As a result of these problems, the presented system is trained from scratch, as very few LMs exist in a pre-trained form to deal with short sequences with visual grounding like captions. CLIP is a baseline, but there is little experimental evidence that the visual embedding space shares topography between languages.

It is also especially evident when considering the magnitude of existing work that the required hardware and scale, using web-scale data, 1000 +GPUHours per run and collaborative teams, are not a suitable goal for the remit of a single thesis. Optimisations are needed given very limited run times: minimise the model overhead and loss overhead of using a metric like BERTScore. Running BERTScore on any

meaningful dataset is beyond most run-time capabilities in the university. Running it to govern training in an RL approach is almost unthinkable!

This work also projects all languages through the same encoder, assuming them to be the same, and thus that the tokenisation schemes and embedding spaces suitably model a set of all languages. Shared embedding spaces can be problematic in languages that do not share scripts or grammatical syntaxes, as exhibited in the Thai language or Chinese, where the typical word separation on whitespace rarely applies.

In the landscape of training enhancements for CLIP, several studies have explored the potential of gradient caching to mitigate the constraints imposed by limited hardware resources. These methods primarily focus on optimising the use of available memory and computational power by storing gradients from previous mini-batches and reusing them in subsequent training steps. This approach aims to increase the effective batch size without requiring proportional increases in hardware capacity. Despite these efforts, the scalability of such techniques often faces significant challenges. Issues such as the staleness of cached gradients and the overhead of managing cached data can degrade training efficiency and model performance, particularly in complex downstream tasks. Therefore, while gradient caching presents a promising avenue for hardware optimisation, its practical implementation requires careful consideration of these trade-offs to truly enhance the training process of models like CLIP. They still suffer from the same scaling issues that are present around limited data shapes and cannot overcome this difference in hardware and training scales.

After this research was carried out, Llip was presented, by [70] sponsored by FAIR, indicating the advantages that variety in captions presents. The approach they use is similar to that in models like BART [75], Flamingo [4] and others, using cross-attention between caption and visual tokens to improve the image encoder prior to the final contrastive loss. Using this cross attention is an elegant approach to ensuring that the image can be queried by all captions; comparatively, our implementation assumes all captions agree on the visual semantics of the image and focus on the same thing. Llip, on the contrary, shows that variance in captions improves comprehension, and integrating this approach into the training pipeline could be very promising for future endeavors.

The improvement offered by Llip would be high, as the variance between captions is the cause of much instability in regions of the n-dim training: when captions focus on different ideas that may cross into relevance for other in-batch samples. Therefore, by querying each caption against the image first, a latent embedding is created that should be unique, which aligns to a specific visual token in the transformer.

2.2.4 Related Works for Linear Sum Assignment

Numerous attempts exist to create LSA-capable networks Aironi et al. [2] document many of the approaches that have been put forward for mathematically complete approaches (where the input domain is any real number, $\Re$). However, other approaches have been trialled for approximation using neural networks, such as bidirectional LSTMs, allowing rows and columns to be aware of each other as shown by Nguyen and Kim [97] offering good approximation results, with the added positive that being gradient-based approaches, they can carry a gradient, allowing the inputs (outputs from other models) to be trained to a specific assignment. However, other approaches look to use the computational graph itself for this differentiation, similar to the differentiation used in methods like gumbel softmax, where an assignment can be differentiated as if a softmax output. This is seen in graph approaches as proposed by Aironi et al. [1]. The most interesting aspect in approaching the problem as a DNN is the assumption that neurons can implicitly learn the assignment as a function of their activations as offered by Lee et al. [72]: not dissimilar from the assertion of VAE approaches elsewhere in this work. These paint a picture of numerous architectures used to implement LSA approximations; all are acceleration-capable, and my function of being neural networks is intrinsically capable of gradient carrying.

From these works, there is a clear benchmark of precision 70%. This is not to be confused with the assignment score. The score totals the selected assignments, where the precision reflects whether the row of the permutation matrix is correct. A precision score of 80% does not mean an 80% score. This is a very powerful approach for evaluating against ground truth: it is more sensitive to teach the optimum and only the optimum solution.

The precision is mathematically defined as the operation

$$\alpha = \frac{tr(Y_{gt}^T)}{tr((Y_{gt})^T \cdot Y_{gt})}$$

where tr$(\cdot)$ returns the trace of a given matrix, and $\cdot$ denotes a matrix multiplication. The result is a similar measure to the graph produced in Figure 6.4.

The significant drawback is that there are a large count of matrix configurations that can have multiple solutions, precision is indifferent to the assignment score, and can therefore be unfairly low for an equal score. In further tests, continuous values are used to avoid these conflicts, with enough tests used so that any introduced error is statistically insignificant.

For use as a metric during training, Y is transposed so that the smallest dimension is the last. This ensures that, for the loss, every column has an assignment. Retrospectively, an auxiliary metric is used, which, based on similar metrics, will be called the Recall. Recall performs the same operation on the transpose of the

matrix, where columns or rows can have null assignments. Recall therefore measures the algorithm's ability to assign only the values it should.

For training a DNN, the formula for α is modified to offer just the confusion matrix of:

$$L = CELoss(Y^T Y^{gt}, I)$$

In this formula, I represents an Identity matrix, signifying each row and column only matches itself.

Using CELoss here rather than the CELoss on the logits directly, is inspired by the prior use of contrastive training, with the hope of a persistent gradient that is row and column sensitive compared to doing an operation as $L = CELoss(Y, Y^{gt})$ The latter approach is curiously preferred in literature alongside a set of inputs that take only integer values with L2 normalisation and loss, giving Precision as an evaluation metric.

2.3 Exploring the Robustness of Embeddings

The term 'robustness' in the literature tends to approximately mean that the embeddings or encodings encapsulate a sufficient set of properties to perform well in many down-stream tasks. Quite counterintuitively, the resultant set of tasks invariably descends to linear probes. Even at conception, CLIP was tested by linear regressions at various depths to ensure distinctions. This demonstrates the power of high-dimensional (512) feature spaces in which linear transformations can be applied to solve myriad tasks. This shows that properties are preserved, but not always that they can be expressly targeted other than finding tasks for which a certain property or knowledge is uniquely required.

For this reason, the following tests are devised using standard datasets (COCO, VisGenome) to expressly evaluate CLIP, and ultimately, versions trained within this work on specific knowledge. This is used for modeling as the best way to provide a broad, but small dataset that appears in the literature despite the work to show the importance of contrast and variance [125].

The questions answered by this are several, with myriad applications:

- How effectively do CLIP encoders predict class entities from the text in a supervised manner rather than top-k retrieval?
- How effectively does CLIP verify knowledge triples of entity-relation-entity in a cross-modal manner?
- Can CLIP predict object masks within an image, and which encoding is best?

- Can CLIP embeddings perform relation extraction from images where multiple objects exist?

Unsurprisingly, all of these tasks seek explicit formalized grammars for the entity relation tuples. Whilst useful for verification, it misses the point of such complex approaches. Tools like DETIC, which utilize 2 stage detectors to semantically label objectness, are more significant for their open-set capability. Indeed, many works focusing on relation extraction emphasize a very specific scale - training sets often have objects with similar sizes in relation to whole images. It is almost impossible to label everything in an image. It would be ludicrous for an annotator to select each grain of hay in a haystack; there are so many that the act is almost meaningless. However, part of the core understanding needed to isolate the visual object of a 'haystack' is knowing that it is made up of many component parts. There are several approaches to help encoder models like CLIP isolate such bits of information, such as contrastive counterfactuals and linear probes. Linear probes are much stronger and have applications in many computer vision spaces, such as dealing with adversarial attacks [91].

Nevertheless, trialling these approaches is key to understanding the style of semantics embodied within the geometry of CLIP embeddings in 512-space.

2.3.1 Mask Prediction from 512 Space

Black-box thinking often assumes that if the model is capable of a function and humans would break these tasks into subfunctions, then a model must be capable of the subfunction too. This glosses over the complexities and assertions beneath approximations with models.

Such thinking can be dangerous. In the case of CLIP, if the function of the model is considered as querying the contents of an image, a subfunction might be sensing a fixed element at a location within the image. Such thinking would be to assert that if CLIP can query images for objects, surely the locations are knowable. Practically speaking, the probability bounds produced are insufficient for a production model based solely on the CLIP architecture.

Beyond the well-established need for global embeddings from early works like GloVE [101] Works like DALLE ([28]), DALLE2 ([73]), stable diffusion methods, and especially NERF approaches ([38, 123]) all point to extreme capabilities of the embeddings learnt from contrastive training of encoders to encapsulate semantic information including localisation to specific object geometry. Latent embeddings are especially key in conveying semantic ideas that often span novel token sets. A common misconception is that 512 spaces simply encode detection information within an image. To highlight that this is not how it is working, and CLIP instead has a good semantic knowledge, it can be demonstrated that the embeddings do not encapsulate

the information of entity masks. This means that instead of training these methods are not rebuilding an image but rather understanding the gist of an image and capturing salient semantics while understanding the role and expectation of forms within the image.

The following experiment shows that using the loss metrics available to DETIC [138] frameworks to calculate masks using DICE loss, does not meaningfully converge on CLIP - partially because the areas in question are so large in the image that masking is meaningless.

DICE loss is used because it is one of the few loss metrics available for pixel-by-pixel loss.

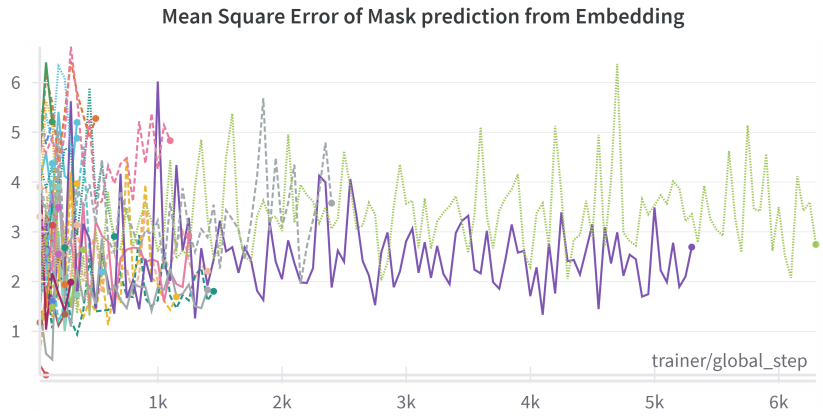


Figure 2.7: This plot demonstrates a hyperparameter sweep that reflects the problems that arise from assuming that semantic embeddings carry visual information. A transformer architecture does not converge in a hyperparameter sweep to being able to predict a rough mask from a caption, across many varied sizes, shapes and other hyperparameters

This figure shows the convergence down to a plateau in most cases: printed samples show full mask cover indicating the issue with pixel-wise prediction: each pixel path through the network receives the same input. As a theoretical experiment, this is where diffusion-based methodologies triumph by introducing sufficient noise levels that neighboring pixels are disambiguated.

Figures 2.9 and 2.10 show how the embeddings from Images and Captions are able to predict the masks: both were trialled because ostensibly they result in the same value in the embedding space (or at least within a linear projection of each other). They show that both are very unstable and result in high mean error. The error in each case is remarkably similar to the fraction of the image covered by the annotated mask which is consistent with the generated images' masks being full. A key heuristic

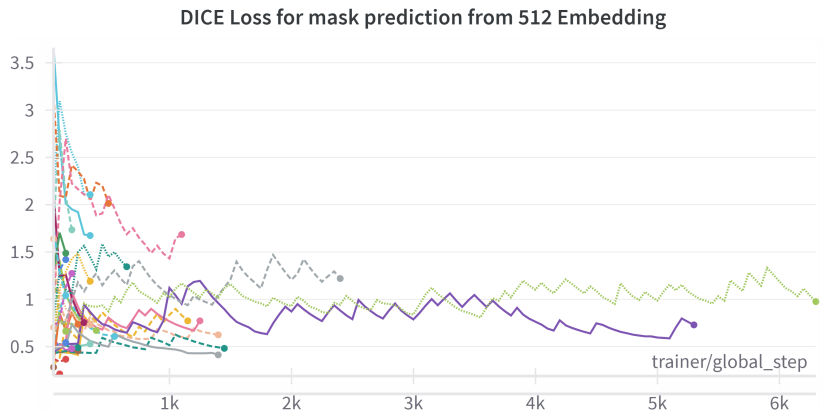


Figure 2.8: A Graph plotting a hyper parameter sweep of transformer models rebuilding object masks from CLIP prompts. Notably, no models converged and the loss is unstable.

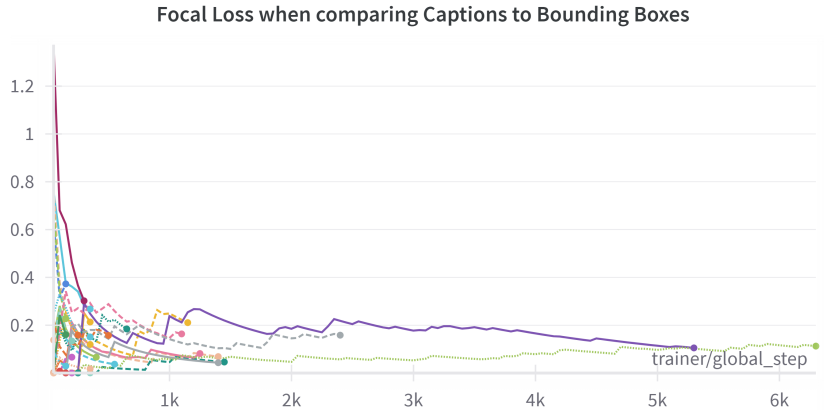


Figure 2.9: This plot shows the error when predicting bounding boxes from captions, the fluctuations are caused by an unpredictable number of boxes between images and therefore batches, and not converging across a hyperparameter sweep

to consider when looking at any training, is stability. If the assumption that the image and caption embeddings were similar held true, it can be expected that they are both equal predictors of mask. In an attempt to put a numerical value on this balance, a weighting factor is used for both inputs' respective losses: $(\alpha \times L_1) + ((1-\alpha) \times L_2) = L$.

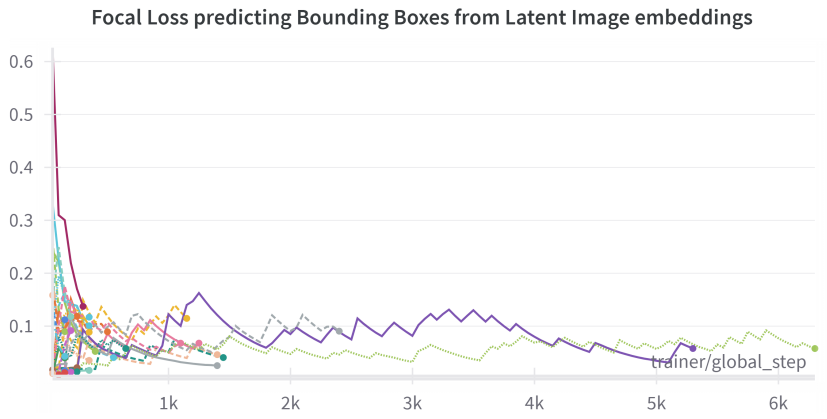


Figure 2.10: A graph showing the loss when predicting boxes from the image embedding across a hyperparameter sweep, showing the same fluctuations and issues as the model using captions, which is indicative of a very well trained underlying CLIP model, with very well aligned captions and images within the VisGenome dataset

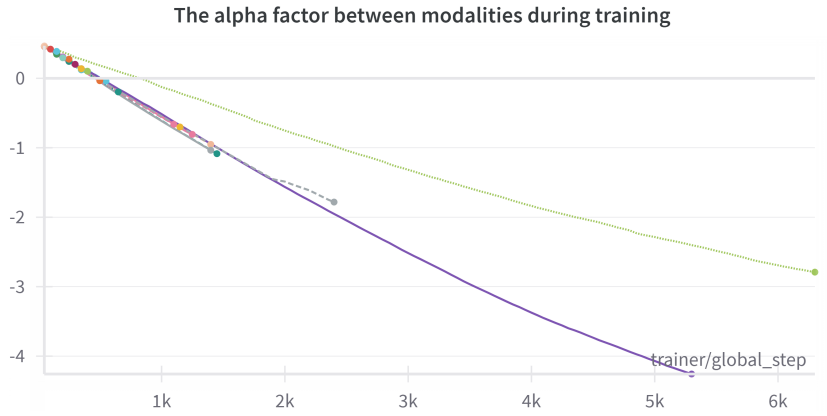


Figure 2.11: This graph plots the alpha factor which balances the prediction based on visual and textual domain. The negative value across all hyperparameters reflects training instability

2.3.2 Relation Extraction

For Visual Relation Extraction, the CLIP embeddings are interrogated using annotations known to be in the image to indicate whether they can either generate or validate predicted entities. The underlying test here is that models like DETR naturally use the generated embeddings to govern class prediction, though this is

initially demonstrated using object names (and other nouns in general).

Given that CLIP is famous for capturing semantic information in a way hitherto unheard of. A rudimentary experiment is performed using the data in the Visual Genome dataset [68] which annotates a proportion of the visual relations present in an image. During data collation,bounding boxes are aggregated and basic NLP tricks are used to encapsulate the annotation in a pseudo-natural language sequence to generate a CLIP embedding, then pass the aggregated box and caption to CLIP to be fed into the DETR model [64, 139] to learn using the masks generated from the DETIC model to train from.

The following set of graphs is the result. The loss graphs show a large number of

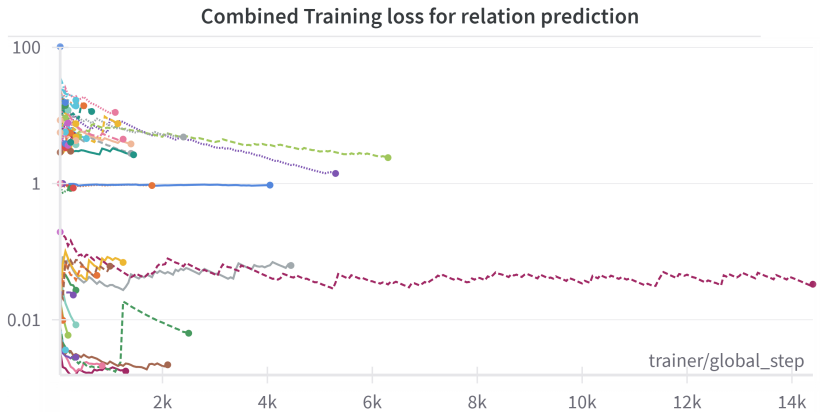


Figure 2.12: The VRE Loss here is derived from the combined bounding box and caption generated by combining boxes predicted by DETIC and generating a prompt to combine items. That the model does not converge indicates that the visual semantic does not contain information on any single prevalent visual entities. This graph has a logarithmic scale because several hyperparameters tried involve scaling the losses being agregated across orders of magnitude

unstable training runs: even though the loss slightly improves, which is inconsistent with the published training profile, indicating that the embeddings are not queriable by a stage detector in this way.

Contrastive loss in this setting is a means of making the model parameters converge in a more stable manner. PairDETR [60] uses a pair of encoders which each use the backbone features as inputs. The assumption is that in using contrastive loss between the produced logits, it forces the salient features to be emphasized and encourages the 2 encoders to learn from each other during training. The reason why 2 encoders are used is to ensure they capture complementary features and are often fed annotations that differ on where they center on an object. Figure 2.13 compares the logits that

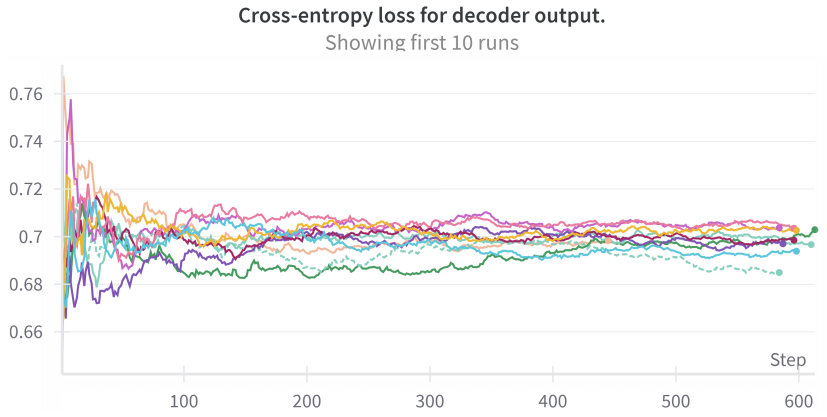


Figure 2.13: A graph showing that pair-DETR frameworks contrastive loss for relations not converging when given a long-distance relation, they are lower because of the architecture of pair-DETR uses the centre of the image for comparison on the final encoder, allowing each to have separate, complimentary losses

are decomposed into box localization. The difference in how the loss is calculated is why the latter is significantly more stable. The DETR variant model fails to converge meaningfully here due to two main factors: firstly, 2-stage detectors prioritize objectness before labeling, filtering out boxes with multiple objects, which hinders the model's function; secondly, CLIP embeddings are trained with weak image-level supervision, differing from the target domain of smaller sub-image box predictions.

2.3.3 Object Detection Using 2-stage Detectors

From the literature underneath two stage detectors [33], it is clear that this is a practical approach to many computer vision applications [35]: the exact nature and interactions of an object are meaningless if all that's needed is to avoid it. Figure 2.16 shows a prototypical training step for the tested pair-DETR [60] framework. Many individual parts of the image are isolated and (mostly) correctly labelled, but the relations are not detected. The nouns show some awareness of purpose, but it is unclear whether that is intentional or just a remnant of verbose labelling. In this image, an understanding of why relation extraction, annotating, evaluation, or comparison is a difficult task is demonstrated in this image: from both an annotation and algorithmic standpoint. The reader is invited here to consider every pair of items and how their relation might be annotated, described, and listed in an efficient data structure. During this task, it will become apparent how interconnected, redundant, and intricate many of the relations are. This observation governs the latter chapters,

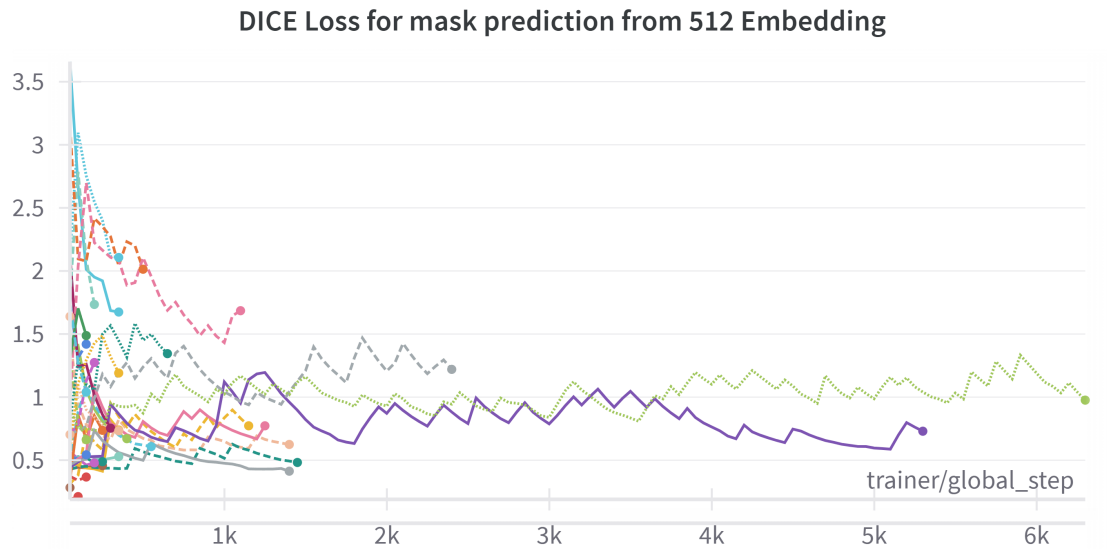


Figure 2.14: DICE loss during training. DICE is calculated with a Cross Entropy Loss at a pixel level

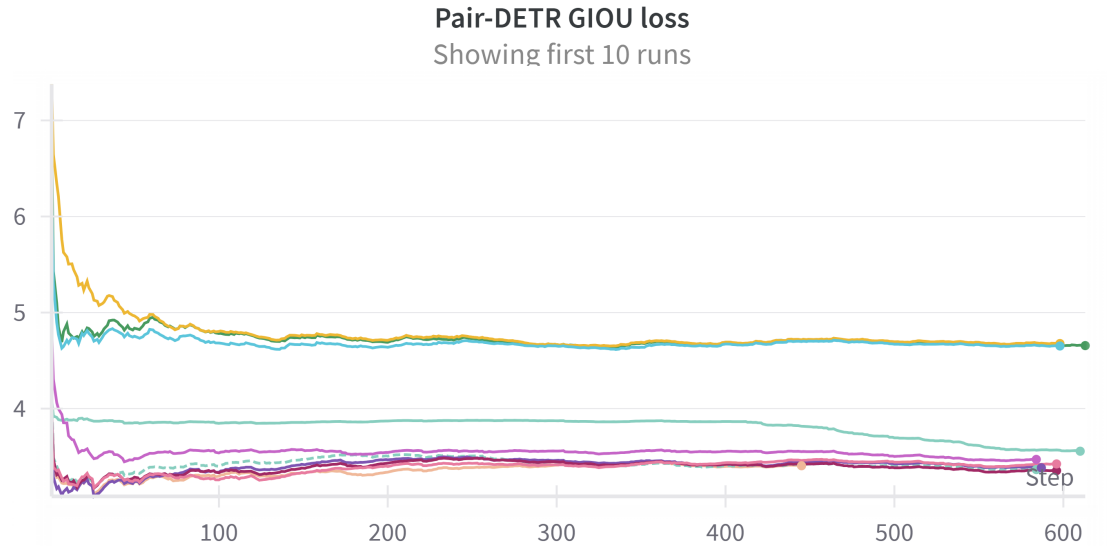


Figure 2.15: This graph best demonstrates the weakness of PairDETR, that the bounding boxes do not correctly converge onto relations of items

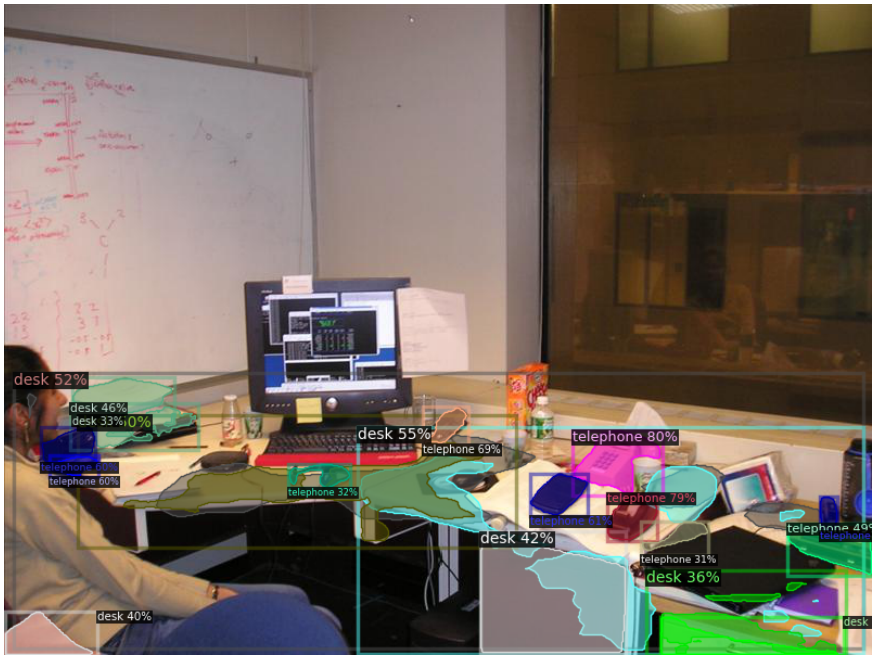


Figure 2.16: This figure shows the annotations generated by DETIC, which relies on a set of engineered prompts through the CLIP model is excellent at predicting segmentations for entities with high 'objectness'. This is the tool used to generate captions and bounding boxes that combine pairs of objects

emphasising encoder systems that encapsulate such information without requiring such information to be annotated so verbosely.

2.3.4 Conclusions

Thus far, a lot has been said on how CLIP works, and the similarities to other works in the NLP space are that the advancement that CLIP makes over prior work is the implicit inclusion of in-batch negatives in an unsupervised manner.

Against the background of semiotics, it is clear that this offers the underlying semantic embedding space access to 2 + orders of knowledge. An ability to group entries such as separate from other clusters.

In an academic context, there are many issues with this methodology.

1. Researchers cannot replicate models at this scale
2. The quantity of data is unreasonable for any researcher to verify, interrogate or compete with
3. Academic institutions are reliant on a private company for insight and access

Therefore, replication should be a significant priority. This work hypothesises that the extreme efficacy of CLIP in both vision and language domains comes from these additional orders of knowledge, and that the scaling of this with batch size is due to the multiple logits per item in each training step.

We therefore seek to exploit this scaling with the addition of extra dimensions.

2.3.5 Section Acknowledgements

The benchmark and demonstration experiments in this section were greatly aided by the provision of hardware, time, supervision, and equipment by the SPARC 2023 workshop.

Chapter 3

Using Additional Dimensions for Teaching with Visual Grounding

In this chapter, the focus is on **RO.1** and **RQ.1**:

RO.1 Present a training paradigm that is not tied to a domain prediction but can create meaningful embeddings. **RQ.1** Demonstrate where and how elements of knowledge can be isolated within a machine learning model.

3.1 Teaching an Encoder with Visual Grounding

3.1.1 Objective

The goal of this work is to train a visually grounded language model encoder using a data source comparable to that available for many low-resource languages.

3.1.2 Goals

- The work will show that it gains a reasonable understanding of a toy language compared to an MLM training approach.
- Catalogue the scaling implications of new methods on the accepted scaling paradigms of CLIP.

3.1.3 Hypothesis

The work produced by CLIP was revolutionary in the creation of zero-shot performance encapsulated in a single embedding space. Therefore, this work proposes the use of CLIP encoders of both image and language to train a tertiary encoder.

There are also merits to fine-tuning pre-trained encoders with these additional languages, as much research has latterly been focused on the differences in reasoning and world-view that is inherent to different philological views; however, it is likely that this would require a magnitude of knowledge, by definition, not available to characteristically low resource languages. The idea of training two encoders to replicate CLIP training must be quickly discarded despite its effectiveness. Training cannot be replicated on academic scales of hardware and data.

Therefore, our experiments will focus on the impact of training on the tertiary encoder, infusing teacher-student training with an additional teacher to instil visual grounding. The hypothesis is that by obeying the scaling available in the contrastive method, a model can be trained with visual grounding requiring significantly fewer samples.

3.1.4 Experiment Definition

To aid understanding of a toy language space by adding visual grounding. By replicating the success of CLIP's text encoder at a smaller scale, the usefulness of this work to low-resource domains will be demonstrated. Given CLIP's ability to use large-scale noisy data, this experiment shows a proof-of-concept architecture for learning from social media where images have noisy captions and may feature incomplete input sets.

To achieve this goal, a text encoder will be trained alongside the pretrained CLIP Vision encoder, allowing the text to be supervised both by the comparative learning of the text components but enforcing an embedding space to carry the constraints of visual semantics.

The test will include 3 encoders. 2 will be pre-trained, and the final encoder will be trained from scratch, converting an unseen language to CLIP-space as guided by the 2 pretrained CLIP encoders.

A benchmark will be an MLM training approach. The encoder-decoder model will be trained with the same masked captions as the original model with the same number of iterations.

Comparison will be performed by freezing the weights to the encoders and training a decoder to the encoder.

3.1.5 Method

This experiment is going to add an extra encoder, as shown in Figure 3.1. A contrastive loss consisting of 3 sets of logits is used. The goal is to maximize the case where the i-th item in each set matches. By exploring different combinations of 3 sets of logits, there is a comparison of how each scales comparatively to the

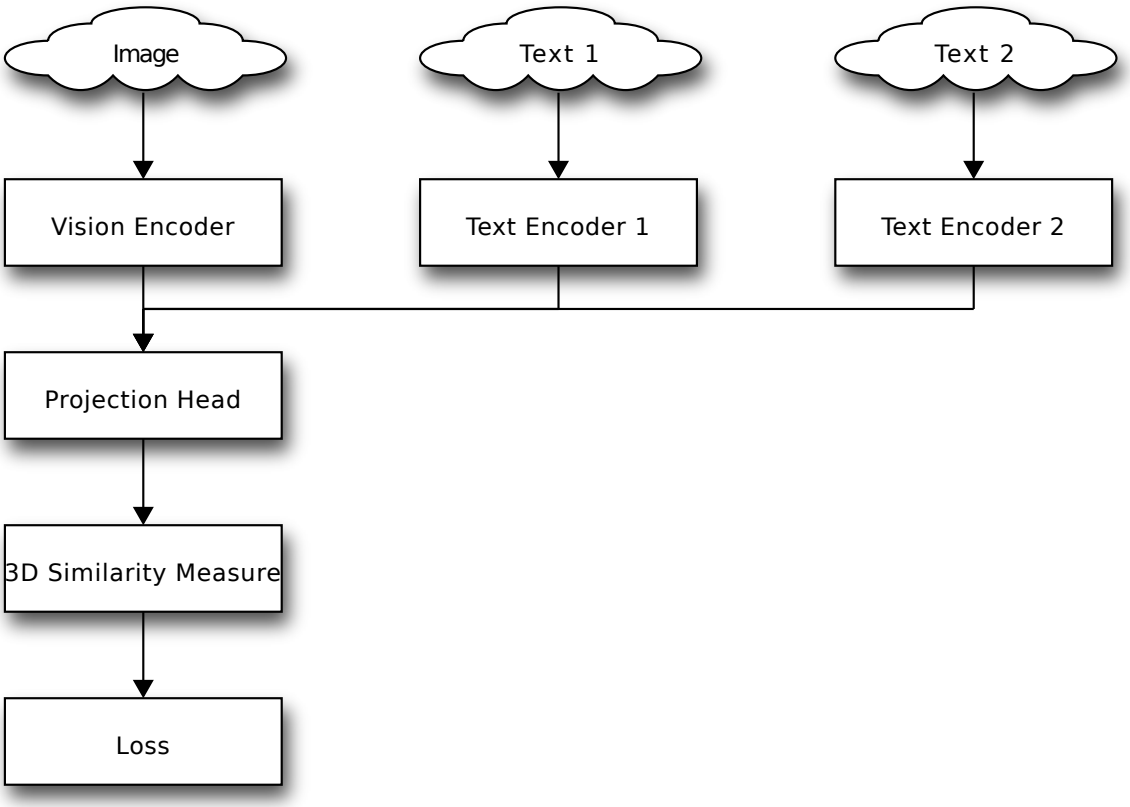


Figure 3.1: The flow of forward gradient during training. An image, accompanied by 2 captions is encoded, and then a 3D similarity measure used.

CLIP model. The merit of this case is to allow for a hybrid capability where, in some instances, 2 modalities may match and be trained accordingly. Positive and negative logits are monitored to show any significant variance in bad logits, showing the conflicting cases.

3.1.6 Dataset

Retrospectively, teaching an encoder a language is difficult from such a limited sample as a few hundred captions [10]. Therefore, the exact choice of low-resource language here is a secondary concern to the work as a proof of concept. To demonstrate the work in this chapter, both a toy Spanish dataset generated using MSCOCOES [42] and some MSCOCOCN [78] datasets are trialled. The latter being applied because it has human annotations from another cultural perspective, rather than machine generated language. This makes the MSCOCOCN a far more fitting choice to tie back to the **RO.1**. For future experiments and exploring the limits of these methods in subsequent chapters, this work defaults to English as a toy language due to available data. It is hoped that this work will aid in calls for data in more interesting shapes.

For emulation of a few thousand samples that may be the upper limit of any low-resource language, the splits of MSCOCO are taken. This gives short-form sentences that each reference an image. The choice of the five captions gives the added benefit of emulating a slightly larger language choice available to each encoder but is still highly comparable to the quantity of gold-standard annotations in many low-resource languages.

3.1.7 Logit comparison

There are many problems with the 3-way generation of a similarity matrix as is described and glossed over in Figure 3.1. Deeper analysis of the problem description can be found Mander, Piao, et al. [88]. However, it is sufficient to consider the issues surrounding scaling cosine similarity to more than 2 vectors. In this chapter, and the next, new algorithms are presented that are computationally viable at small scale and handle multiple terms better.

This work presents numerous different approaches to compare logits through a contrastive method, while attempting to optimize the scaling affordances that are present.

Each method can be found later in this work along with approximation functions and other methods that can be efficient within more limited circumstances.

However, fundamental to the method here is the assumption that CLIP, UNITER [19], and other referenced VL models excel over traditional NLP approaches due to the inclusion of in-batch negatives in a way that many prior NLP works aim to include

with manual supervision, which classifies rather than clusters points according to a label: a subtle difference - but it has a large impact where attention is applied to a series of representations.

3.1.8 Results

In initial tests, the loss curve descends quickly and in line with what is found with CLIP, within the space of just a couple of hours on a single GPU. The graphs demonstrate how the training of an encoder in a fine-tuned setting produces a set of embeddings that carries the properties of the encoder model, even though the activation function that may be associated with the gradient of the space is different.

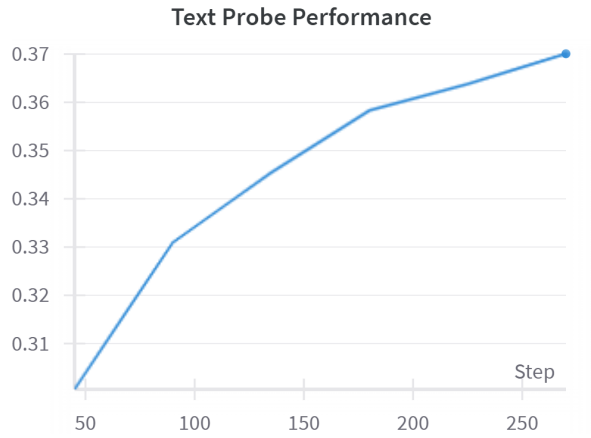


Figure 3.2: Using the Linear Probes discussed in the original tuning of CLIP, and in later chapters, this graph shows how effectively the embeddings in our model improve. This shows that from early in training the probes perform well, and their performance is improved by training, even over a short period.

3.1.9 Sweep results

The wider training report across a sweep shows some interesting emergent behaviors which demonstrate the limitations of the approach. In the subsequent results, training runs with unfrozen pre-trained encoders are included, and runs where no pretraining has occurred. It is reasonable to hypothesize that given the orders of magnitude extra training the pre-trained model had and the comparatively reduced size (indicating model distillation), the untrained model will not converge during our training. This is the contributing factor in the runs that have a linear probe score that stays static at a value around 0.2.

Figure 3.3: This graph shows how training loss descends to minimal values very quickly. Our proof of concept demonstrates the ability for this training to rapidly tune language models with a pretrained visual encoder.Tracking the loss during training shows when the model has mostly converged. Intermittent peaks demonstrates where the model is not generalising well, so notable that the magnitude of peaks also diminishes

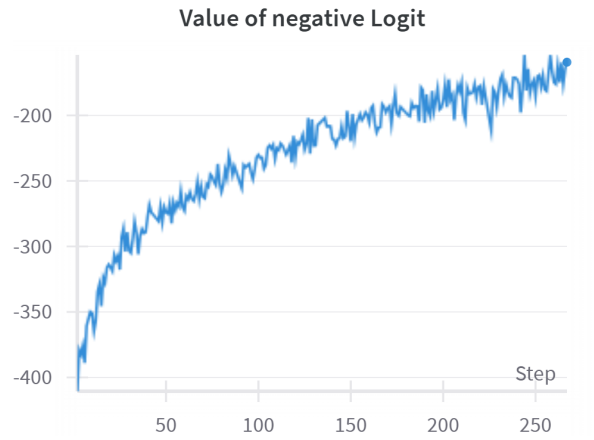


Figure 3.4: Tracking negative logit value during Training to be minimised relative to the positive value. The plateauing of the curve shows the model quickly converging in < 200 steps

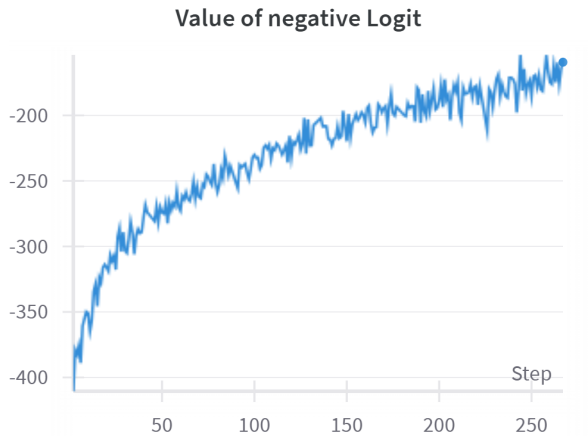


Figure 3.5: Negative logit value during Training

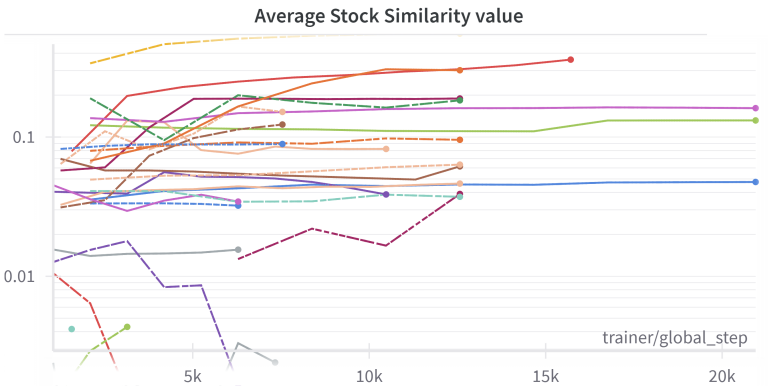


Figure 3.6: This graph shows that the similarity values mean is relatively stable throughout training, across many different hyperparameters. The plateauing indicates the logits that are being maximized are commenserate to those being minimized, indicating a stable training paradigm

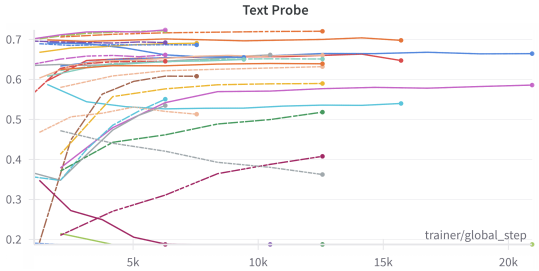


Figure 3.7: This graph shows how many runs converge to a very high classification score for text probes against class, which has been synthetically generated to a top score is unlikely. Descending runs reflect the sweep parameters where a high learning rate is used, or an untrained visual encoder

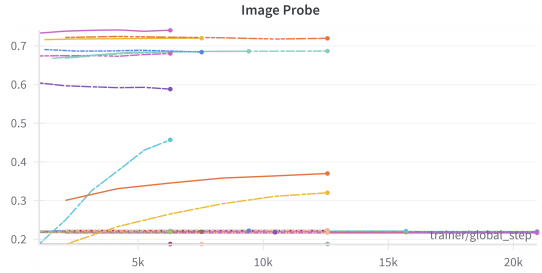


Figure 3.8: This graph shows the training of a Image-based linear probe, from similarity with 3 inputs, notable some of the runs from an untrained visual model grow and are runs where the text probe initially drops in performance. This indicates that both models need to learn together to create to optimum latent distribution

Figure 3.7 shows how a caption can be matched to an instance within the image. Albeit a trivial task for most NLP models against COCO captions, this is zero-shot. In particular, many models have very high performance in 5k steps! Some arrive even earlier.

Compared to textual probes, the vision encoder is significantly slower to converge, where some are immediately high performers, while others take many more training steps. There are several factors that dictate this behaviour. First, only a single loss permutation influences the visual domain; there are fewer unique samples to train from, and there is no pre-trained domain teacher in the combination of encoders. This suggests that image probes will be a better indicator of performance for domains with reduced data or annotation. As the classification is a reduction of the COCO instance annotations, even rudimentary NLP approaches would score well, while curating a visual embedding that encapsulates the semantic information is a far more complex task. Grounding in the visual domain over the textual domain is the dominant metric to estimate performance. To confirm the system's accuracy by decreasing similarity for incorrect cases and increasing for correct ones, Figure 3.6 includes a heuristic: mean similarity per batch. This plot ensures logits remain stable and align with their respective metrics. In 2 dimensions, this is simply the mean of the dot-products of each possible pairing. This should remain relatively stable as the values deviate, but as the number of batched inputs increases, it is expected the ratio of different behaviours to cause increased instability in the mean throughout training. Figure

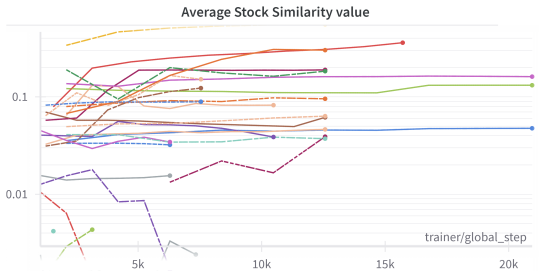


Figure 3.9: Within the validation set, this graph plots the mean similarity score which tracks how stable the training is. The stability of the score varies by both similarity metric, and can be adversely affected by other extreme values in the hyperparameters (like a learning rate that's too high)

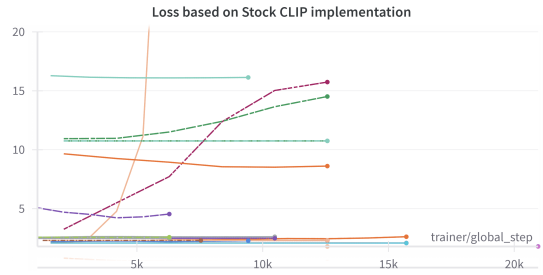


Figure 3.10: Using the cosine similarity we compare our latent space to the pretrained vision model. There are many out-lying runs where the loss increases reflecting a different set of embeddings. Many runs however, are stable with a loss ≤ 1, indicating good training in a majority of cases

3.10 is used for tracking how the overall set of logits is behaving per algorithm, and we can immediately see that whilst the majority are stable in early training, some of the algorithms trialled move the logits to a positive or negative value very quickly, suggesting the models do not converge to the same space, but rather a similarly semantically charged space that requires other metrics to investigate.

What is notable is that the mean projection values in Figure 3.10 remain fixed, indicating a minor, if any, change when there are pre-trained encoders. The projection is set as a parameter to adjust during training. This shows that the projection isn't doing much, if any, of the heavy lifting around semantics, nor is it being used to correct encoder values, when multiple may be combining. If it were becoming extremely positive or negative, it would reflect training instability, or a mathematical issue with the output of an encoder, which is perhaps already being removed by the presence of normalisation, activations, and soft-maxes during penultimate modules. This shows the instability in some algorithms, where the value can suddenly jump, drop, or plunge during training, reflecting that in such a small sample set, minor changes to topology can have a severe impact, especially when considering the working of attention mechanisms. The blue line in this graph and its changing direction is a symptom of the at least B^2 ratio of (maximized : minimized) logits, which first incentivizes the reduction of all logits, before some can be maximized. Performance against stock cosine similarity is also a good metric to consider. In both cases, the validation loss did not converge, which reinforces the hypothesis of other training graphs that the algorithms for measuring the comparative distance of 3 vectors do not

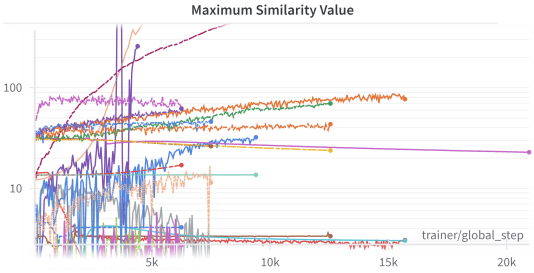


Figure 3.11: This graphs the logit at the 0-th position in each dimension, which should always be maximised compared to the others.

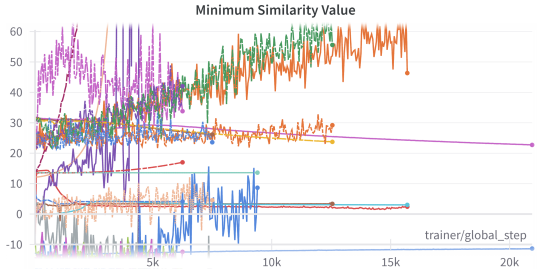


Figure 3.12: This graphs the logit at the worst position in each dimension, which should always be minimised compared to the others.

behave equivalently to cosine similarity and are not transferable in their embedding geometry. It seems derivative of the lack of supervision or label in VAE-style training. In particular, apart from a few standout poor cases, no individual parameter directly

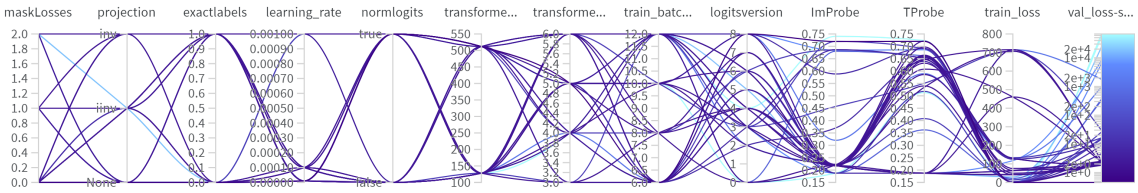


Figure 3.13: The effect of hyperparameters on the output in 3 dimensional training.

correlates to significant or poor performance. During the compilation of this report, other variables were also included, which were analogous to the version of the code, ensuring that any changes in the code did not have a significant effect on training, either causing significantly better or worse training, which would be indicative of errors in methodology. As version indicators were even less significant than minor changes, such as small increments in batch size or Learning Rate, it is assumed that version revisions have had no substantial impact on training effectiveness. The unexpected conclusion to this work is that the many algorithms that are defined in this thesis for n-dimensional vector comparison are not substantially distinct from each other in behaviour, and produce largely comparable results in this training paradigm. With more data, later chapters will explore whether the difference may become significant, but full training and evaluation are beyond the remit and capabilities afforded at this scale.

3.1.10 Findings

The experiments so far have demonstrated highly successful teacher-student training of low-resource languages when paired with a shared embedding space, even when alternate metrics are used. This section has demonstrated the viability of using contrastive training and answered the **RQ.1**, demonstrating the effectiveness of training low-resource languages contrastively.

However, many translation models on HuggingFace and other libraries are not like the encoder previously trained. They have a decoder to allow for sequence-to-sequence translation. The subsequent section attempts to enhance the method presented so far to include a decoder for sequence-to-sequence training and evaluation.

3.2 Training a Whole Model with Visual Grounding

The previous experiment highlights the merit of using multiple teachers with an additional modality. However, in only teaching an encoder, decoder training is further required, which, with a lack of training data, presents many challenges. The question is therefore whether an entire model can be trained effectively in this method.

The significant goal is to add the decoding of embeddings in training. The goal can be achieved easily because there is already a teacher model that exists to translate the target language into embeddings. So the forward pass diagram looks as follows:

A translation model - a basic transformer - which translates to English, then goes through a pre-trained encoder, alongside the pre-trained encoder of CLIP and image encoder. This still requires the same input tuple of the low-resource-language, English and an image. But training will be for an entire translation model. The assumption here is to use the extra dimensions to enforce the model learning embeddings with multiple orders of knowledge by having in-batch negatives that scale better than the 1:B scaling of CLIP.

3.2.1 Tokenization

When introducing new models in NLP, it is important to consider how language is broken into tokens. As previously looked over, CLIP uses a bespoke tokenizer to address issues around locating the [EOT] token, which is used to generate a summarisation of semantics at the end of a sequence. To do this efficiently, they take the maximum token in the sequence, which defaults to the BOS token if not found, which is usually a good subsequent candidate for summarisation. In any case, this unique scheme is unlikely to comply with a translation model. This is why a **marianmt** MariaNMT model is chosen, a translation model with customisable embedding sizes at either end.

Under the hood, MarianMT uses a single linear layer to generate token logits (probably adequate for sentence/word piece tokenization schemes but less likely to perform well with CLIP).

Continuation of the gradient between a translation model and the pre-trained encoder is not available in the standard models. Most off-the-shelf encoders take token ids, and the initial stage of the encoder passes the ids through an embedding space. The nature of this embedding lookup loses any associated backward gradient.

This work requires the assertion of equivalence between linear layers with respect to the transpose of weights of embedding layers. This is demonstrably true in Pytorch.

This allows the conversion of a CLIP text encoder to take a one-hot vector as input rather than a character index. In turn, output of the final layers of the translation

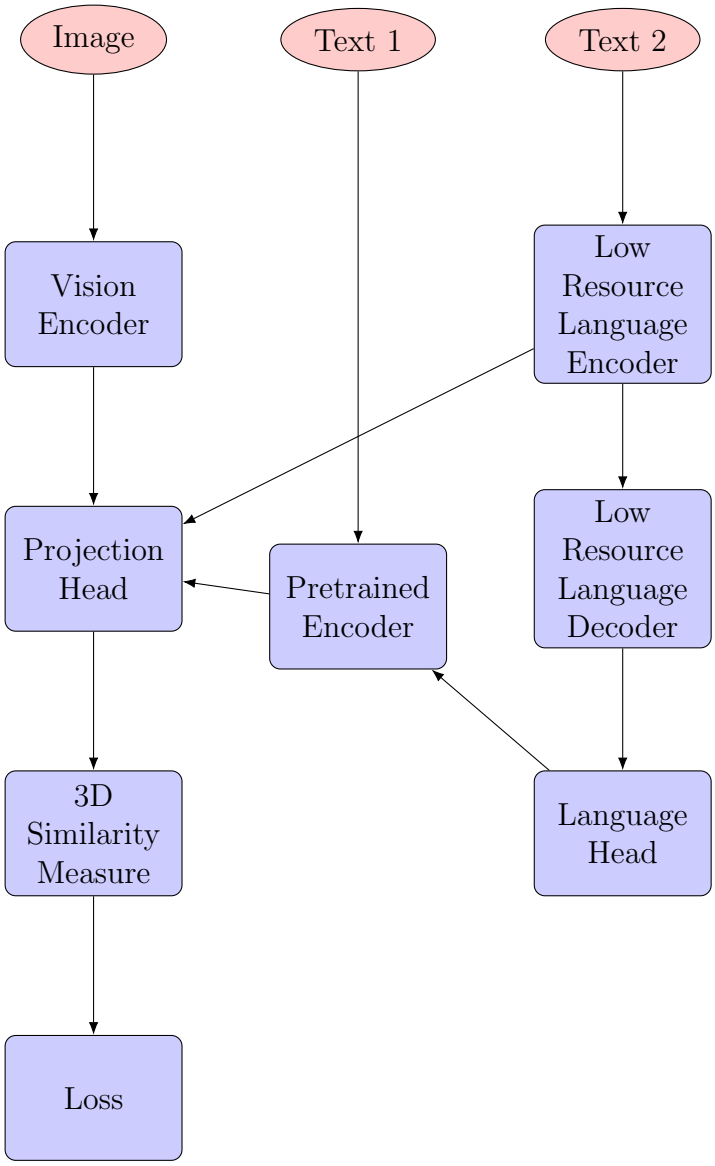


Figure 3.14: A diagram showing the forward pass for full model training

```
function encode_text(text):

    # Step 1: Apply token embedding to the input text
    x = token_embedding(text)

    # Step 2: Add positional embedding to the token embeddings
    x = x + positional_embedding

    ...

    # Step 3: Select elements from x based on the maximum token index in each sequence
    selected_indices = argmax(text, dimension=-1)
    x = x[range(0, number_of_rows(x)), selected_indices]

    # Step 4: Return the final encoded text
    return x
```

Figure 3.15: Code excerpt from CLIP (converted to psuedo code)

```
    function encode_translated_text(text):

    # Step 1: Convert the input text to one-hot encoding and apply token embedding weights
    x = one_hot_encoding(text) * token_embedding_weights

    # Step 2: Add positional embedding to the one-hot encoded embeddings
    x = x + positional_embedding

    ...

    # Step 3: Select elements from x based on the maximum token index in each sequence
    selected_indices = argmax(text, dimension=-1)
    x = x[range(0, number_of_rows(x)), selected_indices]

    # Step 4: Return the final encoded text
    return x
```

Figure 3.16: The change to an encoder model to recieve decoder outputs

model is modified to, instead of taking an argmax of probabilities, convert them accordingly.

There are 3 options for this design choice:

The ultimate softmax activation in the translation model could enable the translation model to forward partial tokens to the encoder; however, as this is the human-interpretable output of the trained model, it seems more prudent to force a token choice where possible. Unlike the usual beam search methodology used in many LLMs. It can be asserted that the CLIP encoder has a good enough distribution to enforce a similar fullness of our embedding space. Initial experiments with this direct conversion of probabilities had very promising loss curves, but failed to produce any intelligible training output. A similar result is found when models exclusively train on generated content [41] or why prompt engineering is difficult, where models attempt to maximize a given output at the expense of legibility.

An alternate solution of a Gumbel softmax layer, which probabilistically converts probabilities to a one-hot output. Enabling the linear layer to exactly approximate the embedding layer while receiving an input according to the probabilities on the final layer of the translation model.

Format	**Tensor Shape**	**Advantages**	**Disadvantages**
No language head or embedding	N/a	This allows clean separation of the pretrained and trained model with minimal overhead	There is no human interpretability of the output. The output domain of the translation model being trained is identical to the context of the pre-trained model and the model will be suitably biased to this domain.
Linear Layer	BxSxD 10×80×512	This approach minimises additional consumption of VRAM whilst allowing a linear fit of the output of one model to the other. There could be some interpretability if the linear weights are topologically comparable to the CLIP or other domain appropriate token embedding model.	The disadvantage to this approach is that there is now a linear layer that can obfuscate the teaching of the translation model.
Language head one-hot with activation	one-hot: BxSxV $10 \times 80 \times 60,000$	Using Gumbel softmax to create a human-readable token choice while maintaining gradient calculation	There is a large chunk of gradients lost to low-precision and sparse vectors. Memory consumption means that this cannot be run at present scale.

Table 3.1: Comparison of different intermediate token gradient conversion

Given that the penultimate translation model layers have a resulting tensor size of (B, S, D) where B,S,D represent batch, sequence length, and embedding size, respectively, and the post-embedding layers of CLIP expect a [B,S,D] input, this shortcut can be made, removing the requirement for the additional BV parameters. The issue here is that the output of this model is unconfined to a natural language output - skipping the language head.

The assumption here is that because an architecture can be trained to a behaviour in an MLM paradigm, it will learn the equivalent behaviour in this new paradigm using a fraction of the data. The hope being that the pre-trained language encoder provides a suitably robust gradient for this operation. Because embedding layers can be treated as linear layers with an indexed rather than one-hot input. Taking the final LM-head linear layer, the identity can be used that any number of adjacent linear layers without intermediate activation can be abstracted to a single layer. Thus, splicing is fine for continuing a gradient between models, with the only real question being whether the model state at this location is useful for later training to a token output.

Work showing that the penultimate CLIP layers are still good for linear regression on output is a good indicator that the assumptions posed thus far are correct. The issue arises in the specific scenario of CLIP sequence summarization training, as it relies on the positional identification of the [EOT] token. In a broader context, it necessitates the fitting of a linear layer, which can be easily accomplished using a linear regression probe and a few translations in a validation set.

3.2.2 Proof-of-concept : Locating the EOT token

A proof-of-concept experiment must demonstrate that there are memory-efficient equivalences to a linear layer and the gumbel / argmax and argmax layers to locate the candidate EOT token space. Our assumptions:

- The EOT token is irreplaceable. It must be in the sequence.
- The embedding for it is likely to be very distinctive, which can be checked by finding the comparison of the weight to others.
- The removal of the quantisation of values when converted into tokens will improve the fit of the model.
- If a Linear layer, L, is initialised with the transposed weights of an embedding space, there is an equivalence for a given input, I:

  $$\mathbf{onehot}(I) \cdot \mathbf{Embedding}_{\mathbf{weight}} = \mathbf{Embedding}(i) = L(\mathbf{onehot}(I))$$

  where $\cdot$ represents a matrix multiplication.

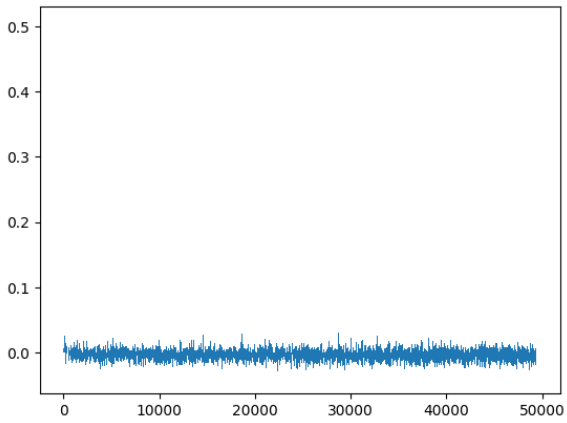


Figure 3.17: A plot of all similaritys to the EOT token by token index, showing each token sequentially along the x-axis, and similarity on the y-axis.

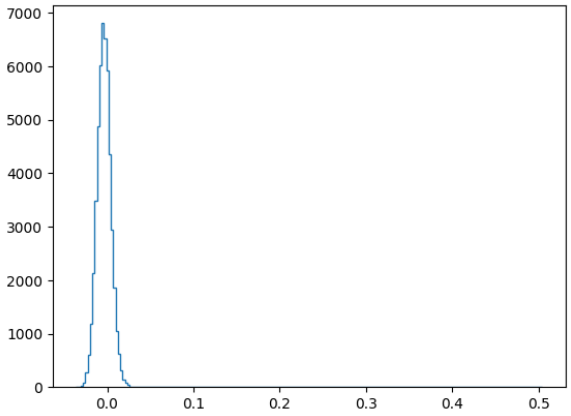


Figure 3.18: A figure showing the distribution of cosine-similarity to the EOT Token.

- Traditional token-loss paradigms insert [EOT] tokens under different strategies as a linear function of the embedding state, which the argmax may not learn, but Gumbel softmax might.

The underlying assumptions above can be quickly plotted by showing the cosine similarity of the token embeddings and the distribution of logits. That nearly all indexes are perpendicular in high-dimensional space, shown by the Figure 3.17, suggests that the EOT token's embedding sits prominently in a single dimension: the hypotenuse of the vector is very close to an axis of the space. This reinforces the hypothesis that the embedding can be used directly to locate the best EOT token while maintaining a gradient.

A similar conclusion can be drawn more clearly from the distribution plot in Figure

```
function EOT_finder(x):

# Step 1: Get the End-Of-Token (EOT) embedding and move it to the device
eot = detach(EOT_embedding).to(device)

# Step 2: Compute the dot product between x and the EOT embedding, then find the index of ↩
    the maximum value
selected_indices = argmax(dot_product(x, eot), dimension=-1)

# Step 3: Select and return elements from x based on the maximum index
return x[range(0, number_of_rows(x)), selected_indices]

function EOT_finder2(x):

# Step 1: Get the End-Of-Token (EOT) embedding and move it to the device
eot = detach(EOT_embedding).to(device)

# Step 2: Apply Gumbel Softmax to the dot product of x and the EOT embedding, then multiply ↩
    by x
gumbel_output = gumbel_softmax(dot_product(x, eot), dimension=-1, hard=True)
x = x * gumbel_output.unsqueeze(-1)

# Step 3: Sum over the first dimension of x and return the result
return sum(x, dimension=1)
```

Figure 3.19: This code excerpt shows the difference between selecting the correct token to evaluate from a pre-trained model, compared to trying to learn the correct positioning of the EOT token. The issue in the latter case is that there is not necessarily a correct position in a semantic-based system.

3.18. There is only one token with a high similarity. Notably, this does not have a similarity score of 1. This is caused by errors introduced in a low-precision setting. (There are optimisations that account for this in training [29]). The salient conclusion here is that the [EOT] token is very distinctive and can therefore be used to query directly by output using cosine similarity and matrix multiplication.
Test Cases:

- Check assumptions - equivalence of weights to input format
- Can I use a pretrained weight to query model output
- Best way to teach the translation model is either gumbel or argmax layers.

Figures **??** show that selecting the closest EOT token is a valid way to find the best location. It may not work from a random initial state where the tokens are not close. Especially as it may lead to a false descent, given CLIP's tokenization scheme where in lieu of the [EOT] token, the [SOT] is used, which may lead the model to very quickly start giving many [SOT] tokens in the same scheme as the default input ids. As such, a gradient descent using Gumbel softmax is used to find the best token, allowing a residual gradient to generalise to which token is best suited to the role.

Training for a specific set of tokens is not ideal, but rather aiming for the generalized rule - thus sufficient quantities of varied data are needed.

In lieu of token-level loss, it is worth considering how the final hidden state of the EOT token is able to adequately summarise the semantics of a sequence: That attention must successfully be used to encapsulate every other token's meaning. However, there are obvious shortcomings to this method, as the ordering of tokens is lost, allowing non-sensical structures and grammars. In practice, the real cause of training difficulties is the lack of beamforming. Beamforming is a technique for generating a sequence of text-tokens with the optimum combination of the next n-tokens such that it maximises the score of the output and each token. Unfortunately, beamforming is often relegated from the training stage of a model because of the complexities involved in preserving a residual gradient without intricate reinforcement learning techniques that would push this technique beyond the scope of this work. But, it means that relatively poor results can be anticipated and expected.

3.2.3 Evaluation Methods

Beyond the methodologies presented in previous sections, the following experiments aiming at sequence-to-sequence translation offer some additional evaluation challenges. First, tokens are in a human-legible form; with such limited sample sizes, finding the optimal assignment for some tokens across a linguistic border is a nuanced issue and is best described as an imbalanced classification task: there is so little data, inferring grammar for long sequence forms with such a limited size is difficult when not using a pretrained decoder.

The tensors used for the training regime presented are all in 512-space, which does not readily convert to an evaluatable form against an input or output of tokens. Therefore, in our test step, a single linear layer is trained according to the MarianMT [66] model for machine translation.

The available training framework and the hardware limits have meant that the loss from the linear layer is aggregated for each batch in the test epoch. This should give a curve at each epoch. This curve is defined by the batch loss minimum (best), maximum (worst), and mean value. BERTScore [133] at the validation and test stage is then calculated in the output throughout the test epoch, as the linear layer is training. This means that the initial values are going to be poor (near random) and the final values are likely to be near perfect. This makes this metric not directly comparable to other literature, but rather it is a good way of combating the overinflatedness of BERTScore, and the tracking of the mean value will reward rapid convergence to an optimum over a fixed number of steps.

Mean Test Loss	5.10
Min Test Loss	1.79
Max Test Loss	10.1
BertScore on Test Batch	0.497

Table 3.2: Benchmark results for initial training. Tracking of the mean loss, shows that in the benchline the model improved linearly with time

3.2.4 Section Findings

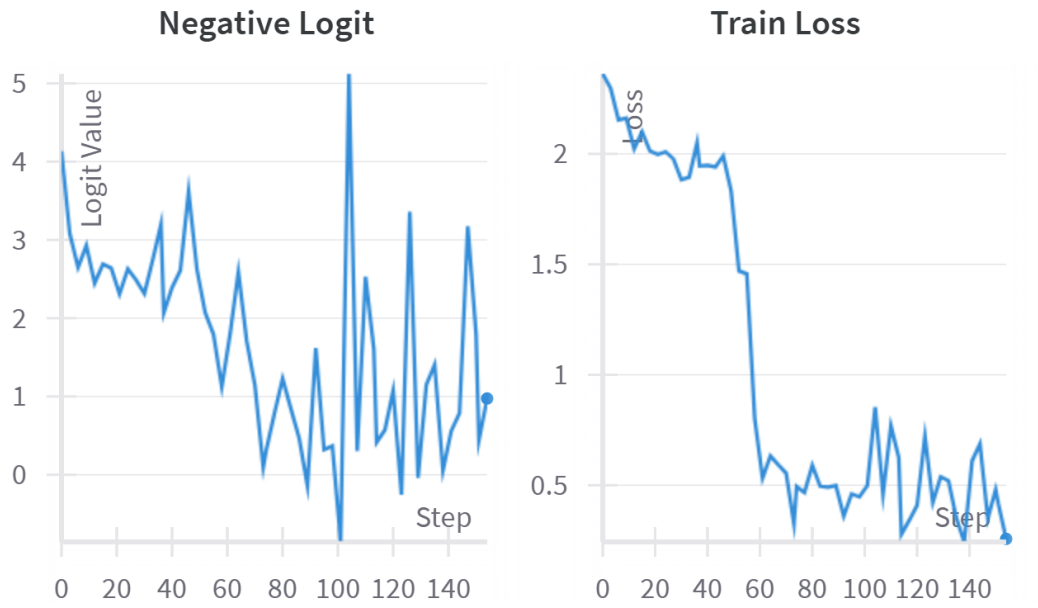
This subsection will explore the results from an initial experiment using the contrastive loss across all pairs of encoder outputs simultaneously. The input to each step is a single image and caption pair from the COCOCN dataset. (So this includes the image embeddings from CLIP; the intermediate translation embeddings and the CLIP encoding of the MarianMT model output). As previously discussed in the earlier Method section, the expectation is that the CLIP text encoding of loss will be sufficient to teach a grammar or structure to the outputs.

The function of this experiment is to determine whether, without a Ground truth annotation, the image encoder is strong enough in the representations to accurately train a language model. If this test is successful, it conveys that the created encodings are of such potency that token sequences' meaning can be inferred by their relation to each other, considering the images' relation to each other.

Importantly, this baseline result shown in Table 3.2 shows that the training of an LM head is not effective at such small epochs and steps. (A BERTScore of 0.5 can be easily achieved by most translation models and even custom weights, as demonstrated in the publication [87]). The log of the mean, minimum, and maximum test loss indicates a roughly linear descent throughout training, though the absolute values are still very high.

The conclusion of this work is that the model didn't adequately converge. Manually inspecting outputs that were incoherent and had repeated words rapidly confirmed this metric-based hypothesis. There are several and myriad prognoses:

- Train the language head slower and better - possible from the pretrained weights of CLIP
- Longer training with more steps to better optimize past the initial descent fluctuations (the spikes at 100,120,140 steps, characteristic of contrastive loss descent with too high LR)
- The relatively high plateau of training loss may also reflect that more layers are needed - this method uses 3, atypically low for a translation model.



(a) Value of the negative Logit (b) Train Loss

Figure 3.20

The training loss in this run also very quickly descends to a point where it is well predicting the optimum assignment, further training is to ensure the space is free from conflict as things are contrastively sorted. The fluctuations in the Bad logit value represent where vectors that should be different are similar. Relative instability after 100 steps is likely a hint at overfitting to the training batch, or that there is an imbalance in the decoder from the translation model. Such emergent behaviour is a logical deduction when comparing this output to the prior experiment, especially considering the anticipated issues passing a gradient between models.

Figure 3.21 has been included to show the engineering that must still go into such approaches for each experiment to maximize available hardware. The loading of a single GPU during training is a good indicator of how efficiently the available compute is used. Throughout this work, the experiments shown are subject to strict compute limits, and many compromises are made for them to run. Ensuring the maximum utilization of available tools is critical to such small scale work. During the baseline, there are a lot of excess latency spikes in loading, which reflects that the loading of data is not well synced with the execution. Drops in utilization, memory allocation, and power usage show that the batch size can be increased to make better use of available hardware. Increasing the batch size has a quadratic impact on the number of logits tracked and the increase in the efficacy of training. This graph is

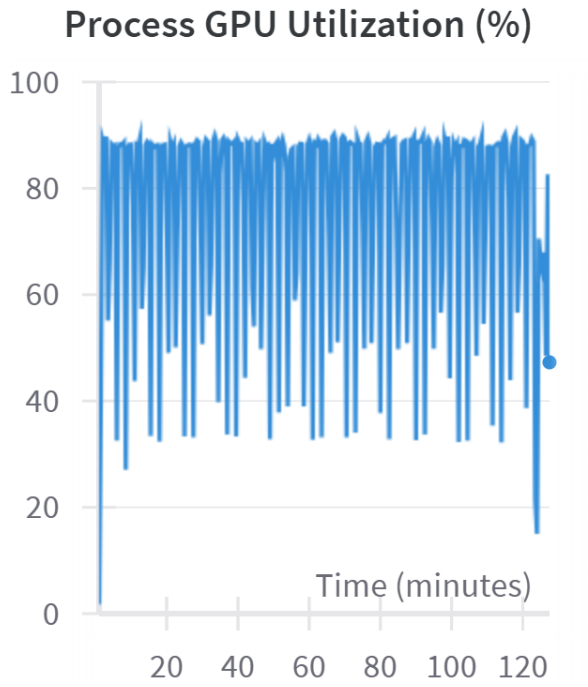


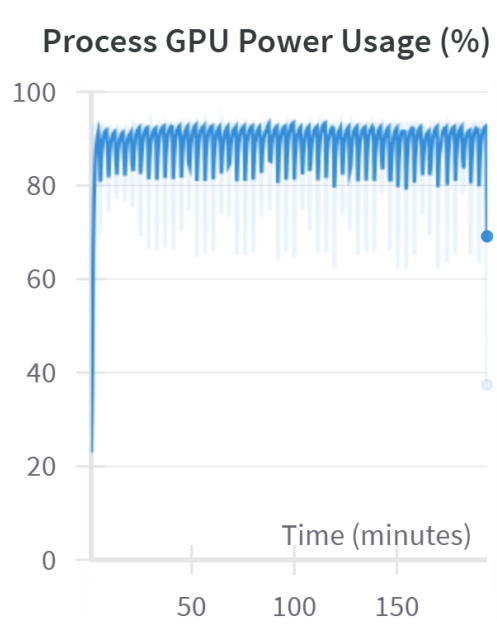
Figure 3.21: GPU loading during training shows very efficient compute making high usage of the GPU, but not completely using hardware indication either memory or compute bottlenecks

generated with a batch size of 14. A batch size of 24 gets an increase of 25% in GPU memory and a utilization near 100%. However, correctly utilizing the accelerator here reduces the number of steps per epoch. An increase in LR and/or Epochs can rectify this.

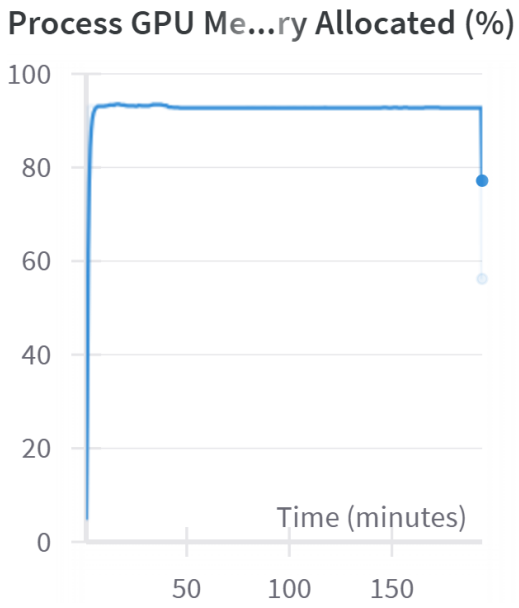
3.2.5 Hyperparameter tuning and Improvements

In subsequent training cycles, the following changes were made:

- Reduce the number of batches to aggregate gradient over. From 16 to 4. This change is to increase the number of steps per data input. This lets the convergence happen quicker in the training time period.
- Increase the layers of the MarianMT model. From 3 to 6. This should have the effect of improving model efficacy
- Increase Batch size to maximum per device. From 14 to 32. Increases efficiency



(a) GPU Power Usage



(b) Memory Allocated During 4D training

Figure 3.22: A Figures showing the final heuristics for compute usage, reflecting more stable and complete utilization of available hardware

of contrastive training, and allows more epochs for a given dataset. This also balances the additional number of steps introduced.

- Increase the learning rate of the language model head. This should significantly improve BERTScore.

Figure 3.22b shows that the model learns very quickly with the associated improvements; however, there is a notable 'elbow' in the training graphs. This is likely caused by an optimum arrangement being found and potentially represents a saturation point to which the data set is fit. It would be logical, therefore, to increase the batch size here to enable greater contrastive comparison within the dataset. Subsequent training from these improvements results in the following figures, showing a marked improvement in utilisation and performance, with the difference between the positive and negative cases (tracked by the positive Logit and negative Logit graphs respectively) only increasing.

Counter to prediction, the increase in batch size has smoothed out the elbow in training loss and moved it from occurring at 60 steps on to around 110 steps even though the monitor for Bad logit value does still show descent at that point. This

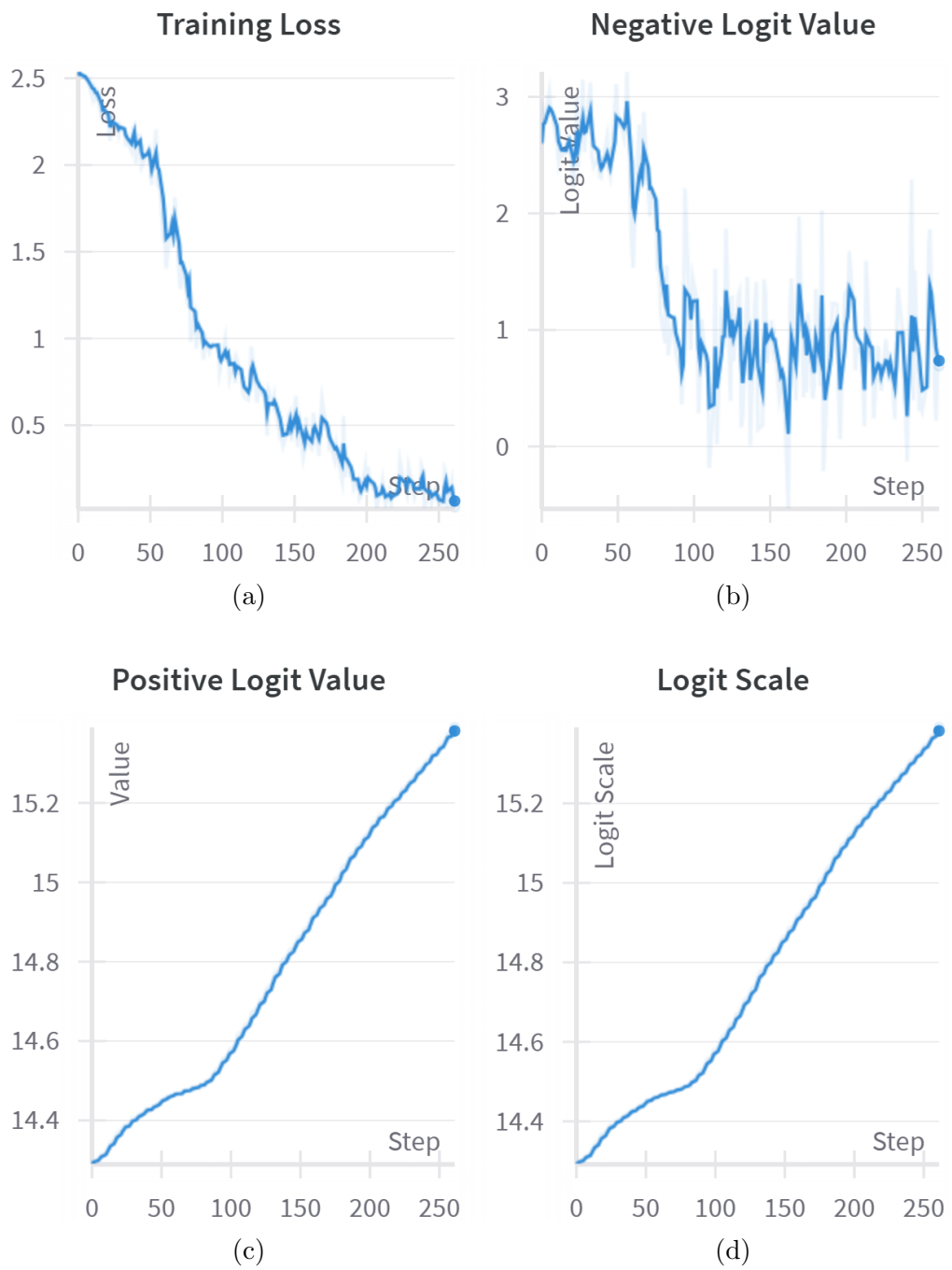
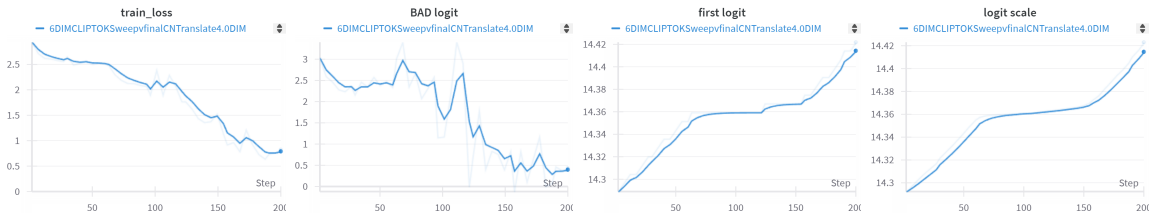

Training Loss
Loss
2.5
2
1.5
1
0.5
Step
0
50
100
150
200
250
(a)
Negative Logit Value
Logit Value
3
2
1
0
Step
0
50
100
150
200
250
(b)
Positive Logit Value
Value
15.2
15
14.8
14.6
14.4
Step
0
50
100
150
200
250
(c)
Logit Scale
Logit Scale
15.2
15
14.8
14.6
14.4
Step
0
50
100
150
200
250
(d)

Figure 3.23

answers **RO.3**, showing that batch size drastically improves training stability and points to the need for an even higher batch size, or at least a comparison of more points during this training.

3.2.6 Final Hyperparameter sweep



(a) (b) (c) (d)

Figure 3.24: Compilation of Final Training Report Figures

Table 3.24 shows the marked improvement in training formats for translation tasks. The logits descend well and quickly learn low-resource languages. It is worth remembering during comparison to other works that each of these occurred at academic scales, in very limited runtimes on single GPUs.

To ensure the validity of this training, the test loss was calculated using cross-entropy loss on the tokens against a human translation, The plots shown in table 3.25 are showing the plots of the min, max and mean values, demonstrating that the models descend very quickly to a near-optimum token prediction case. It is worth understanding that in the core implementation for the marian model, the language head is a single linear layer, which, for the purposes of encoder-only systems, is trained during the training epoch. Therefore, there is some natural error compared to more sophisticated token prediction techniques used in current LLMs. However, simply printing the result of these tokens is not a fair or meaningful comparison in this context. Least because the chosen low-resource language is not one the author is fluent in, but also because at this scale any good or bad samples would be trivial to bias in sampling. Therefore, all datapoints are instead available via the Weights and Biases interface.

When put through a hyperparameter sweep, the results of this methodology for training are a clear indicator that using additional dimensions to contrastive loss is a clear improvement on existing methods and has clear application for training models in low resource languages.

Most interestingly though, for translation-based tasks, the following graphs compare how this occurs in multiple dimensions.

Understanding Figures 3.27 and 3.28 is helped by considering the distribution of values that results in the mean. It is also prudent to note that the trainings were aborted if the value increased. Thus, the mean as a percentage increase in the minimum value reflects how fast the training converged to an optimal solution. Despite promising training heuristics, inspection of "translated" sequences from

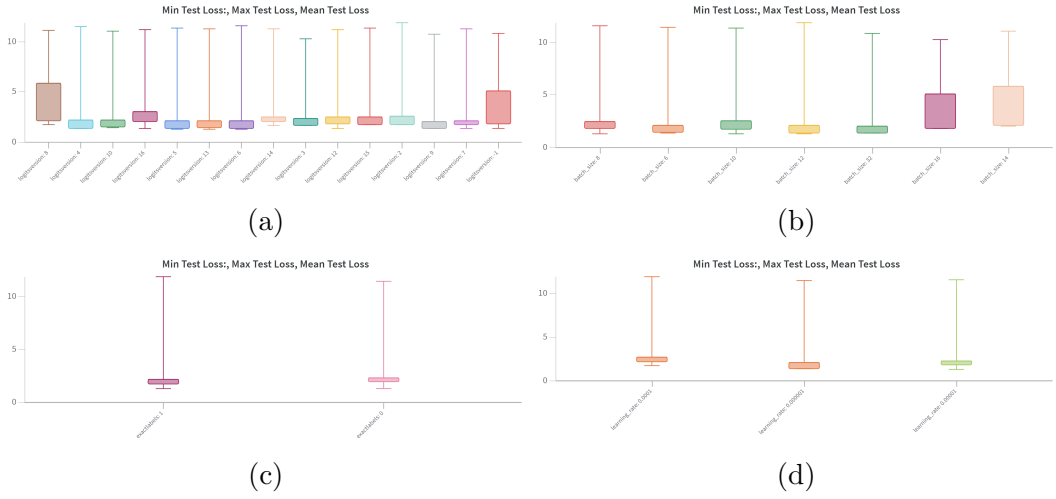


Figure 3.25: Compilation of Final Training Report Figures

Figure 3.26: Comparison of logit method performance in 4D

the encoder demonstrates the problems with unsupervised training at small scale. Following this result, it is apparent that the literature around PGD attacks offers a glimpse of how sensitive the representations that models learn can be, as imperceptible changes can significantly alter even simple classification tasks. In this sense, the same is true in language, where local maxima can occur in token selection, causing sequences of tokens to align improperly. If deployed at scale, this finding may have limited impact, as previously mentioned techniques like beamforming reduce the likelihood of this occurrence.

This highlights that the gradient function and the encoder that emphasizes semantic over token choice struggle to prioritize any single token over others. If token choice were a contributory to the loss function, then argmax would rapidly solve this issue. However, as an intermediate component, the loss of a gradient here would be a critical issue. A way of navigating this choice whilst preserving a gradient is to instead use a Gumbel softmax. This function, typically used for peak prediction in forecasting

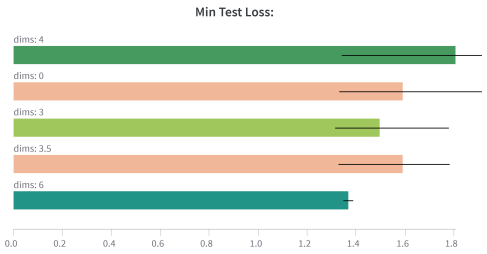


Figure 3.27: Minimum scores during TestLoss grouped by dimensions

Figure 3.28: Mean Test Loss grouped by dimensions

models, allows a probabilistic one-hot selection that maintains gradient; however, in testing, it showed to introduce too much noise for convergence, and ranking parameter importance indicated that not using it led to stronger runs.

3.3 Conclusion on High Dimensional Loss to Guide Translation Tasks

Subscribers to universal language theory would suggest that all language tasks can be split into several subtasks: defining the language space and traversing the space. The theory of a universal language suggests that every language is a distinct subset of all languages. This theory elegantly explains the issues of loan words and native speakers being able to easily interchange languages. It also links nicely to subjective language models as needed for hate speech detection tasks. However, many multimodal models suggest that the visual (or physical domain) is merely a projection away from this domain; naturally, cross-modal guidance from this space can be extrapolated, using learned weights from pre-trained models to calculate gradients.

The idea is that where there are shortcomings in the ability of language models to get varied representations well grounded onto other domains.

In recent years, models like BERT have become prolific for grading generated language and guiding beam searches for generation. The general idea of this is to use a model that will be sensitive to nonsensical tokens. It will have an interpretation of all tokens within the context of a semantic rather than as a set problem. Practically, however, this means feeding the ground truth into BERTScore as tokens, relying on hidden layers and performing a loss metric between the BERT latent representations of the prediction and of the ground truth.

For an initial experiment, this flow is altered to instead take a picture as ground truth, comparing via pre-trained weights the visual encoding and the encoded text. There are several drawbacks to this approach - BERT is semantically sensitive, where a visual grounding is more likely to be sensitive towards practical elements and less so around word order. BERTScore also relies on a multilingual model that is trained and fine-tuned in many domains. Conversely, CLIP (at the time of writing) is suited to just caption-image pairs of web-scraped data, not to long prose that may be found in translation tasks. However, retraining would be possible at a later point.

3.3.1 Future Work

Thus far, it can be seen that having multiple teachers in a student-teacher pipeline can be a very effective way to teach a model, especially in a low-resource context where pre-trained weights can be transferred across domains. Later, this work has not just been modified to receive a low-resource language, but there are also many applications, each of which would be worth their own thesis. One of the big adaptations to this code base is the ability to take truly n-dimensional inputs that can vary across batches: meaning that at inference, different encoders could be used. Historically, this may have been applied to polyglot methods reliant on individual encoders for language families, but

may have more use where style differs in nuanced ways. A primary example would be social media comprehension, where the medium of a post, text, and comments may all be in the same language, but splitting by encoders, even across the OP and comments, would be very sensible due to differing language limitations and stylings.

Chapter 4

Analysis with CKA and Other Methods

Comparing the efficacy of the training methodologies proposed in this work with other methods is not easy. Multiencoder methods, with myriad downstream (and upstream) applications, do not lend themselves to evaluation for lack of first-order knowledge outputs. As discussed in the Background Chapter, first-order knowledge is the direct attribution of a property - the crux of most evaluation metrics. However, as prior work has shown, systems training single encoders exhibit multiple orders of knowledge. Evaluation on datasets that the model has seen is insufficient to know whether the model has encapsulated generalisable knowledge; exhibited by correctly clustering locations, the clusters must be assessed against unseen categories. In this chapter, **RO.2** is achieved: exploring the methods of evaluation available for such models.

A further challenge to an academically rigorous analysis of the models that underpin this work, and the benchmarks used, is that the methods introduced are novel and contemporary to this work. Many mathematical methods have yet to be explored or applied to these models, which have different workings from many other applications, so few apply. It cannot be understated that the model, CLIP, which subsequent chapters are based on, has already got over 20,000 citations since its publication early in the creation of this document. Therefore, blind spots in publications and literature are inevitable, but every effort has been made to ensure the novelty of this work and find any comparative studies.

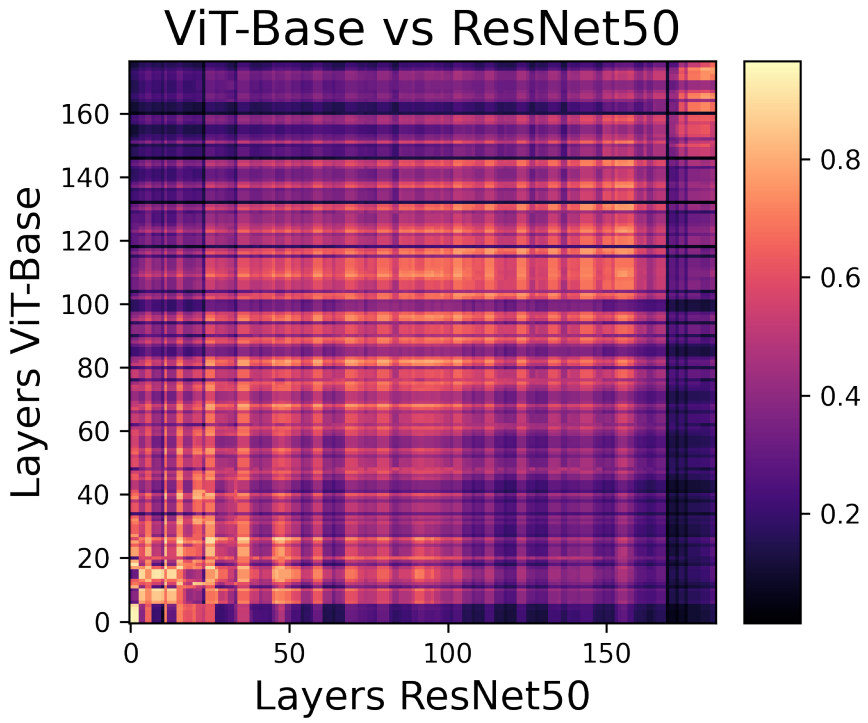


Figure 4.1: This Figure, a replication of prior work, [98], reveals the significance of layers at each depth of a neural network with a specific dataset. The horizontal repitions demonstrate the homogenous properties of intermediate ResNet layers compared to a ViT architecture

4.1 CKA

However, this does not prohibit the benchmarking of this new approach. Our training performed within a day, if resulting in weights similar to other works that take longer, is still a hugely significant contribution, especially when considered in conjunction with movements for cleaner AI. This method is often used to compare models and will be used in this work to compare a random model with one of the known good weights.

Therefore, a comparison is made to investigate the model weights with existing off-the-shelf weights. As a means to compare models, the Centralised Kernel Alignment (CKA) method is commonly used, primarily for comparison between model architectures, showing how layers tend to certain weights.

The original formula from [23] presents CKA, a representation similarity metric that is widely used to understand the representations learnt within neural networks.

Specifically, CKA takes two features X and Y as input and computes normalised similarity (in terms of the Hilbert-Schmidt Independence Criterion (HSIC)) as:

$$CKA(K, L) = \frac{HSIC_0(K, L)}{HSIC_0(K, K) \times HSIC_0(L, L)}$$

However, in this instance, K and L are both 3-dimensional matrices of the layer representations across an entire data set, computed for each layer of the models. For an entire model, storing such vast matrices does not scale well enough to be used meaningfully for low-resource contexts or computationally limited applications.

To optimize for low-memory contexts, there is a derivation to calculate in mini-batches which reduces the storage and computational requirement. Subsequent calculation and evaluation use the equivalency that follows:

$$CKA_{minibatch} = \frac{\frac{1}{k}\sum_{i=1}^{k} HSIC_1(X_i X_i^T, Y_i, Y_i^T)}{\sqrt{\frac{1}{k}\sum_{i=1}^{k} HSIC_1(X_i X_i^T, X_i, X_i^T)} \times \sqrt{\frac{1}{k}\sum_{i=1}^{k} HSIC_1(Y_i Y_i^T, Y_i, Y_i^T)}}$$

This is more computationally efficient because the sum operation can be greedily executed rather than waiting for all components. The equivalence to the notation used elsewhere shows this can be rewritten by removing the linear factor, and sum as an average of the HSIC calculation.

$$CKA_{minibatch} = \frac{HSIC_1(X \bar{@} X^T, Y @ Y^T)}{\sqrt{HSIC_1(X \bar{@} X^T, X @ X^T)} \times \sqrt{HSIC_1(Y \bar{@} Y^T, Y @ Y^T)}}$$

The prior notation, on a fixed k on a given epoch, the terms $\frac{1}{k}$ and $\frac{1}{n(n-3)}$ cancel out of the final CKA formula and can be removed. It must be ensured that a last batch of differing size can be dropped. In actuality, out of a particular dataset comprised of over 10^6 samples, the probability that the final sub-batch represents a sufficient quantity of outliers to expose a discrepancy in model activations sufficient to impact the average calculation is so minute it can be safely ignored.

Here, $HSIC_1$ is defined as:

$$HSIC_1(X, Y) = \frac{1}{n(n-3)}(tr(\bar{X}\bar{Y}) + \frac{1^T \bar{X} 1 1^T \bar{Y} 1}{(n-1)(n-2)} - \frac{2}{n-2} 1^T \bar{X} \bar{Y} 1)$$

It might be worth noting that with some especially large batches, these are important scaling factors to aid with precision, which are not applicable for the encoders being trained at small scale.

```
def _HSIC(K, L):
  N = K.shape[0]
  ones = torch.ones(N, 1)
  result = torch.trace(K @ L)
  result += ((ones.t() @ K @ ones @ ones.t() @ L @ ones) / ((N - 1) * (N - 2))).item()
  result -= ((ones.t() @ K @ L @ ones) * 2 / (N - 2)).item()
  return (1 / (N * (N - 3)) * result).item()
```

```
def _HSICv1(K, L):
    N = K.shape[0]
    result = (torch.sum(K) * torch.sum(L)) / (N - 1)
    result=torch.sub(result,(torch.sum(K @ L) * 2))
    result= result / (N - 2)
    result= result +  torch.trace(K @ L)
    return result
```

Figure 4.2: The figure shows the improved implementation of HSIC used for this work to reduce the computational overhead achieving 100% speed improvement

Despite the concise mathematical notation, it is further optimized via substitution which is possible in the case where $X = Y$ such that the calculation is for $HSIC_1(Y, Y)$ where the return is the average of the matrix Y against itself.

The formula expressed as

$$HSIC_1(X,Y) = tr(\bar{X}\bar{Y}) + \frac{1^T\bar{X}11^T\bar{Y}1}{(n-1)(n-2)} - \frac{2}{n-2}1^T\bar{X}\bar{Y}1$$

The formula for $HSIC_1(X, Y)$ can be rewritten with substitution for the case with a single term, showing how this can be simplified when computed. For this purpose, a new function is written in the released code due to the terms that can be simplified in the following using the sum to notate stepping through the 3-dimensional matrix:

$$HSIC_2(Y,Y)\sum_{i=1}^{k}(tr((Y_i\bar{@}Y_i^T)(Y_i\bar{@}Y_i^T)) + \frac{1^T.(Y_i\bar{@}Y_i^T)11^T(Y_i\bar{@}Y_i^T).1}{(Y_i@Y_i^T)} - \frac{2}{n-2}.1^T(Y_i\bar{@}Y_i^T)(Y_i\bar{@}Y_i^T).1)$$

The original paper used the identity of a matrix operation with an array of 1s as an equivalence with the sum operation, using transposes of the array of ones to preallocate memory and govern the dimension of the sum operation. When this is converted to code, the standard implementation based on the affiliated library is as

Figure 4.2 shows the numerous optimizations made to reduce the overhead of the CKA evaluation metric to be reproducible at a smaller scale. This includes removing the ".item()" calls, cause a summation of a computational graph and costly transfer to CPU memory and Python literals. Removal of the unnecessary creation of additional arrays of 1s is also necessary, which can be problematic to different pipelining techniques, and unstable where GPU memory is fully utilized at a time. The removal of these arrays has an extra optimization benefit of removing costly matrix ops, where sum is more efficient. Matrix ops are mathematically equal but implementations are written assuming multiple dimensions, where sum is a special case.

```
def orig_HSIC(K, L):
    b=torch.sum(torch.sum(K,dim=0)*torch.sum(L,dim=1))
    c=torch.sub(torch.sum(K)*torch.sum(L)/(K.shape[0] - 1), b, alpha=2)
    d=torch.div(c,(K.shape[0] - 2))
    output=torch.add(d,torch.sum(K*L.t()))
    return output
```

Figure 4.3: The final function for calculating the CKA return for minibatches. Emphasis is on a minimal number of optimizable operations. Theoretically, a computational graph may is preserved through the execution but exploitation for learning is a experiment reserved for future work.

```
def ORRIG_HSIC(K, L):
    N = K.shape[0]
    ones = torch.ones(N, 1)
    return torch.add(torch.trace(K @ L),
    (((ones.t()@K@ones@ ones.t()@L@ones) /
    (N-1)) -
    ((ones.t()@K@L@ones) *2))/ (N-2))
```

```
def ORIG_HSICA(K, L):

    N=K.shape[0]
    return torch.add(torch.trace(K@L),
    torch.div(torch.sum(K)*torch.sum(L)/(N-1) ↩
        -
    (torch.sum(K@L)*2),
    (N - 2)))
```

Figure 4.4: These code exerpts show the difference that code optimization makes, both in simplicity and preserving computational graph

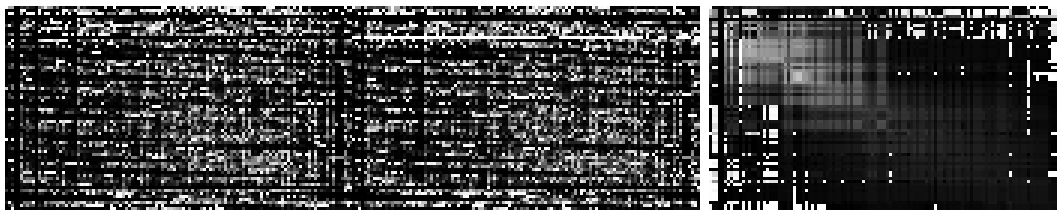
After remedying these errors and removing the final line as previously discussed, the refined version is shown in Figure 4.3.

This modification makes the process almost twice as fast, dropping from 64ms to 38ms on an input of size $(20, 512)$. Though more consistent timing is generated during runs. A further reduction in time is achieved by replacing the trace with an equivalent and faster torch.sum$(K * L.T)$ producing the code seen in Figure 4.3.

Repeated calls to a function, however well written, will always be slow when performed in excess of 50000 times. As such, this code is still suboptimal and modifies the above to facilitate the batched inputs and the additional simplifications available in the special case where $K = L$. It is also notable that, in the original code, the calculations for each HSIC are stacked and summed, but in this code implementation, they are aggregated in place within the respective minibatches first to reduce the memory footprint.

The result of the aforementioned optimisations is a method that can run within the prescribed limits of available hardware, without reducing the number of potential epochs. Figure 4.4 shows the side-by-side comparison for evaluating the output space of CKA and our improved version; notably, the experiments with the improved CKA show computing timing improvements and benchmark any reduction of accuracy resulting from PyTorch's non-determinism.

One of the core improvements that is common to engineering efficient machine learning code is the understanding that array operations are significantly faster than iterating over an array because code can be optimized close to the hardware level.



(a) Modified original method: 6086 Seconds (b) Original method in PyTorch: 6819 seconds (c) Batched method: 346 seconds

Figure 4.5: This figure shows side by side the comparitive differences in the CKA measure between the a replication of the stock code (Centre), the modified code for a single datapoint (Left) and the method that takes a batch (Right). Clipped or null values from extreme dissimilarity result in the white squares - the Figure (c) indicates a more stable computation when results are aggregated by minibatches. Though they each are clearer in showing the relative similarity between early intermediate layers of the trained model which bare no resemblance to later layers of the model.

In MLOps, these optimizations often manifest in how batches of data are collated. The HSIC calculations are such that across all layers the calculation is repeated, and therefore making an extra dimension to each array results in similar speed ups found in batch calculation. Calculating all layers at once, rather than sequentially, is implemented to deploy this code. [1]

The minimal impact of this can be seen in both the precision (through removal of repeated ".item()" calls) and the speed. Using an identical dataset, Figure 4.5 shows the output over a minimal sample of 150 images resized with the standard CLIP preprocessing.

The conclusion from Figure 4.5 is to observe the noise and peak values present in CKA when adding individual steps. PyTorch operations are non-associative. They cannot be guaranteed to have the same output should order be changed, irrespective of theoretical equivalence using algebraic derivations. Without the '.item()' calls, the result is faster, albeit significantly noisier for having more race conditions.

4.1.1 Accuracy of New Methods

Curiously, during the above derivations, an error is introduced. Let the original formula be defined as $\text{HSIC}_0(X, Y)$ and the derivation as $\text{HSIC}_1(X, Y)$. Despite HSIC_1 being equivalent to HSIC_0 such that mathematically (ignoring linear factors) $\text{HSIC}_0(X, Y) = \text{HSIC}_1(X, Y)$, when run, the resultant sum of $\text{HSIC}_0(X, Y) - \text{HSIC}_1(X, Y)$ is occasionally non-zero, taking values around $+ - 1e^{-7}$. A couple

[1] Code available at https://ww.github.com/st7ma784/6DIMCOCO

of theories exist for this. Firstly, the difference in the number of steps at certain precisions creates minuscule errors. More likely, is that in using torch functions, 'torch.div' introduces an epsilon value. This mitigates 'divide by zero' errors, and as the default value is set to $1e^{-8}$, this is trivial to most applications but introduces a minor inequality.

4.1.2 CKA for Model Analysis

4.1.2.1 Hypothesis

CKA and foundational prior work assume a holistic pipeline: gradients calculated from the ground truth data with fixed annotations. The lack of first-order knowledge points or supervision in the training being evaluated presents a novel domain. The training paradigm presented in this work is not widely explored with CKA because the network architectures differ significantly across domains. The neural networks do not geometrically converge to certain output patterns, nor are the comparisons necessarily meaningful between a set of tokens and patches of an image. Therefore, in this section, the question is asked whether CKA is a viable evaluation measure given the aforementioned challenges. The conclusion will inform **RO.2** and subsequent evaluation.

The output of CKA gives insight about where similarity occurs within a pair of models. Testing is performed against a pretrained CLIP model. A model trained with various methods in this work is used as a comparative model. Given each figure is expected to vary as parameters like similarity metric are changed: The hypothesis to test is that there will be 2 observations that are pertinent. The first observation is because of how layers may have to be repeated to increase/reduce the values to yield differing distributions of values in each 512-space vector. These deviations would manifest as seeing horizontal and vertical lines of peak values. Figure 4.6 shows clearly a line of peak values at the bottom and right of the graph. This indicates how the last layer of the CLIP VIT L/14 model has very distinct activations compared to each other layer: enabling a characterisation of the model layers. The upper left corner and lower right also have a different pattern, reflecting comparative differences in these parts of the network, suggesting that the first 11 layers are relatively homogenous, as are the next 11, with the final layers being very distinct.

The second hypothesised observation is that by using an off-the-shelf set of pretrained weights, there is likely to be a stronger set of activations as the model performance increases that correlates with other performance metrics.

4.1.2.2 Methodology

CKA methodology for evaluation can apply to encoder systems; all training is monitored following the derivation in the beginning of Section 4.1 with the batched implementation during validation stages.

To ensure that our hypothesis is appropriate, and demonstrate that unsupervised training can still be monitored with CKA, an ablative study is used. In the ablative study, a random permutation of the features is applied to one of the networks, having the effect of shuffling the orders of activations. The shuffled features show if the method is order sensitive, or its activation must be in the same order to matter. Inserting permutations at each layer showed no geometric constraint. The counter-intuitive nature of this result caused investigation which revealed that in the original methodology, the feature matrix is multiplied by its own transpose. The multiplication produces a consistently $[B \times B]$ shaped matrix, which mitigates this permutation of features; exitingly, this step means that it is easy to modify the workflow to accept tokened inputs by repeating the matrix multiply step for sequential inputs, or flattening, and for networks of differing architectures and modalities, the approach can still apply. For example, a convolutional network with a sliding window over an input with a fixed target will constrain activations to parts of the kernel. Even transformers have significant constraints placed on final layers, with training resulting in nearly one-hot encodings. By contrast, in the case of autoencoders, the inherent lack of geometric constraint during training, except that found by the curation of second-order knowledge, limits the efficacy of CKA as a comparative evaluation with even similar networks.

4.1.2.3 Results

Almost all CLIP experiments result in a plain graph, and the results with the shuffled CKA input show that the problem is actually dimensional order independent. Many of the experiments result in a plain graph. To verify that this is, in fact, correct, intermediate calculations of HSIC and Eigen values were analysed, revealing that CLIP has several activations that are 0, 'inf' or 'nan' that get collected, which have a large, adverse effect on the clarity of CKA measures. One observation from this analysis is that CLIP is trained at a scale where model size is unlikely to be a limiting factor, and the released model has undergone distillation, reducing the number of parameters.

When filtering for these values at every stage, replacing each with a float in the domain (-1,1), produces more meaningful graphs.

To further improve the clarity of these graphs, only MLP and linear layers are monitored, which removes blocks of intermediate layers that get filtered by the nan-to-num process. Figure 4.6 shows the results of the CLIP ViT L/14 model. The

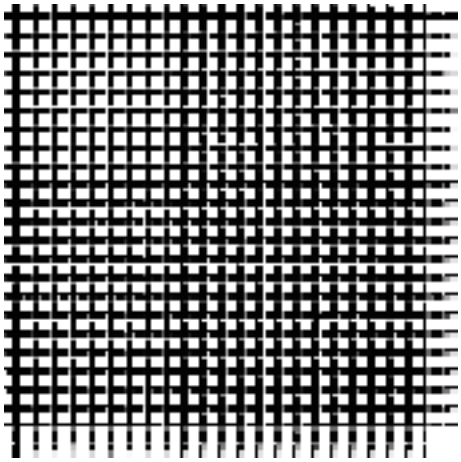


Figure 4.6: This plot shows the activations through CLIP's vision transformer on a single dataset. The start contrast of black and white blocks shows that there is a significant sparse portion of some layers, unactivated by the dataset. The upper right componennt of this figure having a larger white areas demonstrates that the input players of the model are sparser, with richer activations later. There is also a significant change in the final layers, whose activations show a differing structure

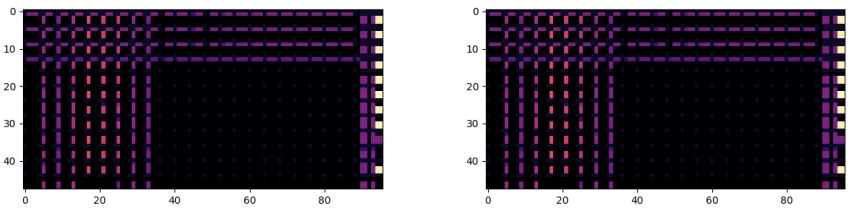
interpretation here is that the distinction between the top left and bottom right parts of the graph shows the unique split in the characteristics of the activations mid-way through the model. This has implications for later work in BERTScore.

4.1.3 Improving CKA on CLIP Models

4.1.3.1 Revised Hypothesis

It is a reasonable expectation then that CLIP-style text encoders are incompatible with CKA methods based on their reliance on the attention mechanisms' effect on a single token, namely the EOT Token. In a traditional CKA approach, all inputs are tested and compared by monitoring all values of the HSIC calculations. In practice, CLIP deviates substantially from other NLP approaches by summarising sequences on the [EOT] token. Consequently, many graphs produced are largely redundant for monitoring posterior layers which have minimal interaction with the latent representation at the EOT location. Therefore, a comparison is made of just the CKA matrix as present at the [EOT] location.

It is therefore reasonable to measure CKA on just the EOT token's representation across the network output. The hypothesis is that this will produce a clearer result.



(a) CLIP pretrained models "ViT-B/16" and "ViT-B/32" with permutations

(b) CLIP pretrained models "ViT-B/16" and "ViT-B/32" without permutations

Figure 4.7: A figure showing that CLIP models are indifferent to permutations of input

4.1.3.2 Initial Results

The results of a plain visual encoder with the CKA analysis over 100 images are the following comparatively blank graphs. In Figure 4.7a suggests that the models are either completely not activating, or that their activations are so sparse that the cross product is overwhelmingly zero.

4.1.4 Further Experiment

Figure 4.7a and Figure 4.7b are generated with random data and exhibit very few high values: It is impossible that for random data, clip does not sufficiently activate. Given the miniscule probability of cosine similarity occurring randomly in high dimensions, as previously outlined in this work, this evaluation method is very sensitive to how effectively the model can understand an input.

Experiments with the input of MSCOCO instead of random, and the limitation of layers to the direct output of MLP components and layer norms (a good snapshot of activations through the transformer) offer the following results, which are cross-referenced to the inputs of the text encoder.

Table 4.1 shows the comparison of 5 different ways of combining model features. Whether comparing the intermediate features of vision; the whole layers of the text encoder; just the embeddings in the EOT position in the text encoder; the projection of those embeddings; or the projection of just the EOT locations.

The Table 4.1 shows the importance of understanding where the model is

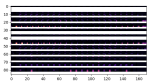
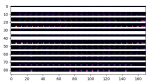
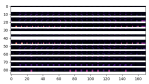
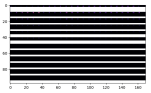
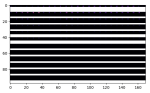
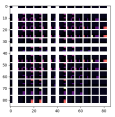
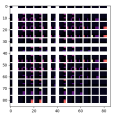
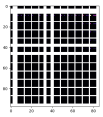
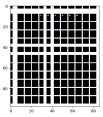
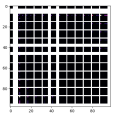
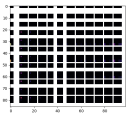
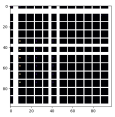
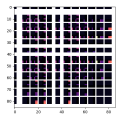
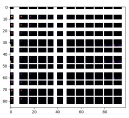
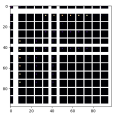
	Vision	**Text**	**Text EOT**	**Text proj**	**Text EOT Proj**
Vision					
Text					
Text EOT					
Text Projection					
Text EOT Projection					

Table 4.1: The CKA activations from different elements of the cross-modal network

activating. The notable shapes to observe in the table are the cross shape, and the location of it compared to Figure 4.6. From both the vision and text components, there is a marked change on either side of layer 10 deep, suggesting that this marks the transition between input comprehension and generating the final layers. This suggests that 10 layers is the upper limit for data saturation even at web scale.

4.1.5 CKA Conclusion

Throughout all subsequent experiments, each trained encoder monitored the CKA during each validation stage. There is a significant optimization of the existing replication libraries to significantly reduce the overheads incurred when using this as a metric for small-scale experiments.

In the small experiments, as shown in Table 4.1, the activations are consistent with the original methods, showing consistent deviations in the model about a third of the way through. Therefore, it can be concluded that CKA can effectively inform the shape of the model's intermediate state, and that the optimisations have been effective at allowing this methodology to be used at a smaller scale. Answering **RQ.1**, the usual performance of CLIP when applied through CKA isolated behaviours and made this algorithm accessible and effective at academic scales.

4.2 Linear Probes

The optimisations of CKA are very effective, but still require expert knowledge for interpretation and do not demonstrate high-order knowledge. They show that the activations are similar, which can be a remarkable indicator among existing models, but cannot evaluate a model for the stated goals of showing high-order knowledge without an existing model.

Another way of exploring the embedding space and affordances of the second-order representations is to use Linear Probes (or Linear Regression Probes). Linear probes have previously been used in many works, including the original CLIP paper, to demonstrate aptitude on a certain line. Essentially, these test second-order knowledge, whether clusters appear together in the embedding space. A linear probe is a fully connected linear layer with sigmoid activation on a single output. This is applied at both varying depths and at various stages of training in the network. Assessment of characteristics like depth, width, and training saturation has all been done this way.

For the experiments implementation, the scikit-learn library is used [100].

Linear probes, [3] are used when a class is known. Generally, prior work assumes that the model training is a supervised learning framework and was designed for computer vision applications as an assessment of class comprehension: ensuring intermediate layers are encapsulating concepts and storing useful features. However, the methodology of many language models is remarkably similar, where the main transformer blocks provide the majority of model functionality which is only converted back to a token-based output at the final minute. Therefore, a linear probe can be viewed as akin to the language head of Marian models used previously - but dynamically applied throughout the model to show that the transformer blocks are learning increasingly effective representations.

In experiments with MSCOCO, emphasis is placed on the ability to classify the image from both a textual and a visual encoder. During the validation epoch, at the start, the classifier is fitted to the stored features. At each step, the loss is calculated and the latest features and labels are stored for the next epoch. Whilst this does mean the validation lags an epoch, in training itself on the last training epoch's features, it minimizes the need to iterate over the dataset, making it feasible within available hardware limits.

Linear probes in this work are not a direct training metric. They show that the embeddings encapsulate useful information, but they do not approximate useful values themselves. Across a number of random runs, the plot of tracking accuracy against the number of steps can be seen in Figure 4.8.

The plateauing of Linear probes, as seen in Figure 4.8, shows how eventually the points stabilize in training, either reaching semantic limits or clustering limits. Semantic limitations can be that distinct clusters are so close together that a linear

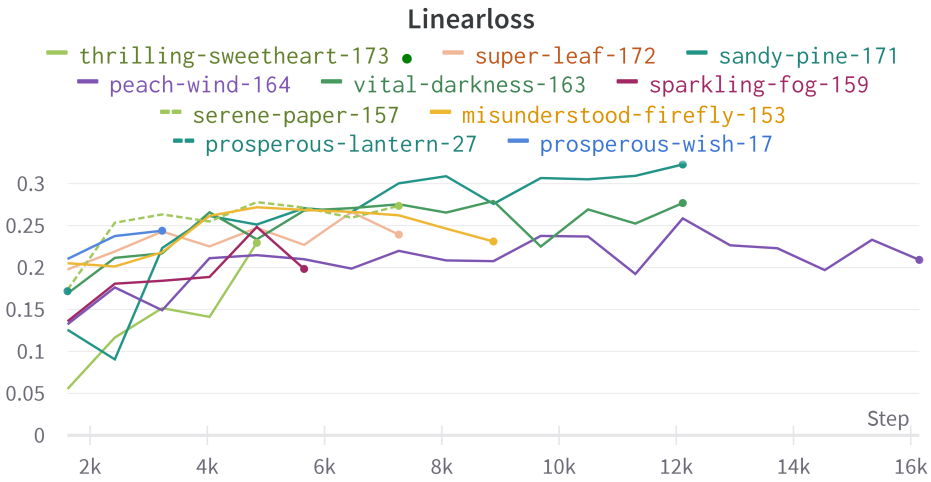


Figure 4.8: This graph shows how a simple MLP improbes through training. The linear probe refits the classes to the projected points, which as training improves, the points becomes more effective

classifier cannot draw a line between them, or that the points do not separate into distinct clusters at all. The latter is unusual and difficult to replicate unless using random values because of the zero-shot properties of the models in question. There is good reason to suspect poor clustering over shape-of-cluster issues: Contrastive training emphasized space between clusters, and the classification labels are taken noisily from MSCOCO: taking the first annotated object, which may not be the salient one, and meaning a single representation could belong to several different classifications. This means it is more likely that a probe cannot linearly fit between 2 classes, which can be considered a clustering limit. Under the experimental limits imposed, CLIP's off-the-shelf performance drops an average of 32.8%.

4.2.1 Zero-shot Linear Probes

One of the core tenets of this work is to prove that extra dimensions are not only a practical way of reducing training overheads and potentially incorporating new inputs. It is also to show the capabilities of the model reflect similar capabilities of the original CLIP model's performance, which are a result of the multi-modal grounding.

A key benchmark used in the paper is zero-shot linear probes, which utilize data not previously used in training. The core conclusion of Figure 4.8, is that even the smallest model can curate a small embedding space and achieve some performance in the problem domain when trained with the classifications given. Therefore, to

test whether a trained model is actually picking up multiple orders of knowledge and correctly learning world knowledge, unseen images are used. The only way they can be clustered and know that a random subset is well represented in the model is if there are layers of parallels as per the prior discussion on orders of knowledge. A good conceptual understanding of the generic objects shown and the semantics, even though they have not been seen, means the model has learned well. Therefore, for evaluation, the imagenet dataset is used: fitting to it using both the text and the image outputs. Using both encoders allows us to test whether each correctly represents the information of that modality (with the knowledge that the image classifications are likely to be noisier)

4.2.2 Using Linear Probes for Small Scale Evaluation

A limit of the number of training steps is needed to balance precision with limitations of available hardware. The number 1000 was selected after observing that most runs have plateaued before that. If there were no limit on practical runtimes, experiments would have been performed testing intermediate layers of the trained models; however, the models are sufficiently small that it is unlikely compared to the original papers that any similar depth analysis would be achieved: There is no point in experimenting on something that cannot be empirically tested.

In our subsequent models, novel methods are tested: The expectation of the theoretical concepts would be a significant jump in performance from prior work. Therefore, the performance profile is unknown. In many cases, this will mean that the linear probes fitted to the model during validation are likely to have an immediate high performance, which at first may appear to be a code error. Those unfamiliar with the training setup may believe that this is erroneous. However, understanding the conceptual leap in the work of [122] where zero-shot models are seen to have their performance rapidly decrease with domain overfitting shows parabolic curves away from optimal domain performance. This is not observed in any experiments: instead, gradually increasing linear probes are seen constantly. This indicates that the contrastive approach is not reaching limits and networks are not data saturated. The limiting factor is therefore the scale of the data used and the domain limitations.

4.3 Evaluating Point Distribution

While linear probes work very effectively at converting an embedding space to a classification problem, it is unclear why an attention-based model would necessarily create points so neatly, especially in open-set conditions. Not to mention that in cases where a model performs poorly, there are multiple critical factors, which are

impossible to disambiguate from the performance metric. It might be more prudent to evaluate against the orders of knowledge expected within the model.

4.3.1 Hypothesis

The aim of this set of approaches is to improve the scaling laws and show that the 2nd and higher orders of knowledge are preserved. During training the approximation of a sentence develops: first, to a location reflecting a basic understanding, and secondly, to a near grouping. For this section, it is hypothesised that the distribution of points is predictably converging as training progresses. An advantage of multiple encoders is the natural plurality of points between modalities. Given each input from a single data sample is semantically correlated (at a minimum) the expectation is that a prominent consequence of 2+ inputs for a modality is multiple, comparable data points. For a given sample, multiple points in the same modality should be closer together than any other. Training could be monitored for this property, but given the direct relation between cosine similarity and proximity, it would be unfair to use this as a direct assessment over the training domain. However, there are other approaches that are not dependent on a redundant input in a modality.

4.3.2 Methodology

To check this, for each iteration of training, validation sets are monitored, checking that the embeddings move consistently, stabilize their values, and then finally ensure points end up near each other.

To achieve successful views of model effectiveness, the following metrics are used:

- The comparative distribution of all embeddings by epochs
- The movement of each embedding per epoch
- Cluster size between points that should converge monitored with both cartesian distance and cosine similarity to demonstrate the model is correctly grouping similar semantics
- The divergence of dissimilar semantics by ensuring that the cosine similarity between points and the cartesian distance is increasing.

During training with einsum estimation, as shown in Figure 4.9, a single vector embedding moves rapidly between consecutive epochs, which isolates performance characteristics without being task-oriented.

Figure 4.9 shows that the embedding value is rather unstable: The amount and direction of the distribution change does not tend to 0. The significant change

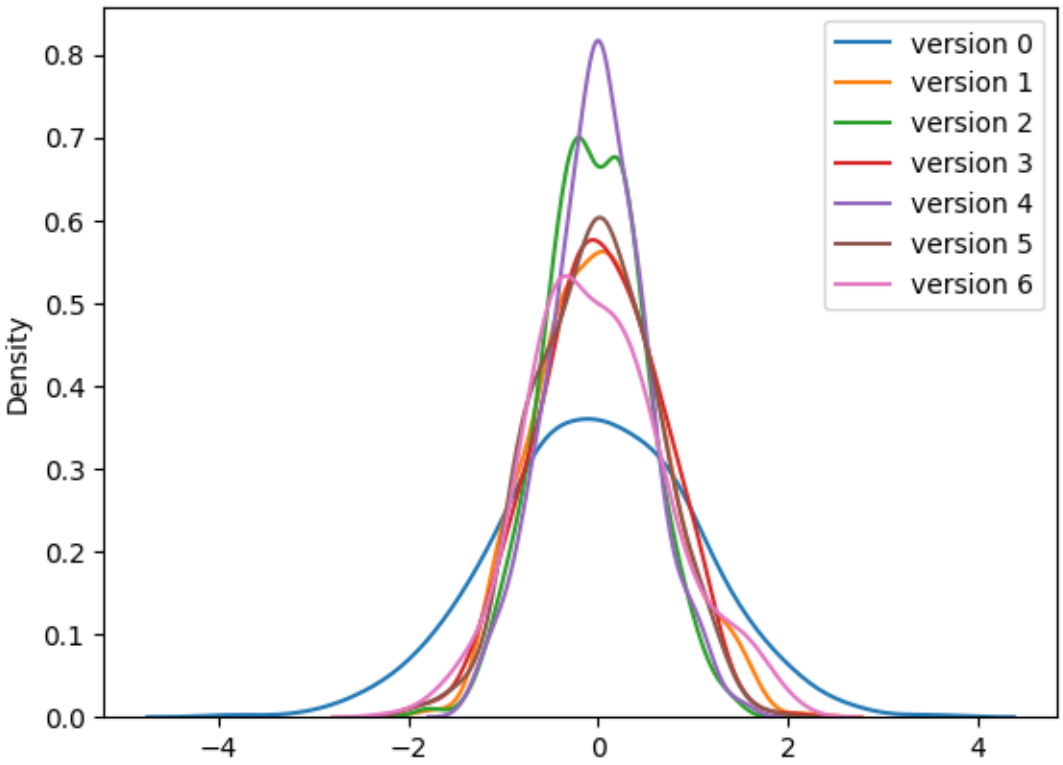


Figure 4.9: This plot shows how each epoch, or 'version' of logits, has a different amount of movement, by tracking the mean cartesian distance between embeddings before and after the gradient step. The plot shows model convergence by each bell curve becoming increasingly close to each other. The vectors are plotted pre-normalisation, to demonstrate the presence of comparatively large values (thus movement can appear large, but have minimal effect on the similarity metric).

each epoch in magnitude indicates instability and significant gradient steps. The behaviour of non-convergence suggests that the similarity measure is poor. From the discussion in latter chapters, einsum has fundamental issues with handling multiples of negative numbers, which affects model convergence. It has been previously shown in the introduction that autoencoders tend towards sparse states, which is likely to be the most unstable for this method as the difference between positive and negative is minimized at 0. From this Figure, therefore, it can be considered a good metric for this training, as a known issue with a particular approximation has been shown as a property of the graph.

It is important to also interpret this graph as a rough indicator of centrality, rather than similarity; it represents the distribution of values in each axis, but not in what order. The rolling delta of the same point between successive epochs is also plotted. The expectation is the same distribution, with a reduction in variance through training. A notable feature of Figure 4.9 is that the lines of adjacent steps are not adjacent, meaning that even the distribution of values often moves and returns, suggesting the need for gradient methods that are less sensitive for individual values, likely caused by the sign errors present in the Einsum approximations.

By observing these relative movements, a measure can be made of how the convergence of a model suitably protects these spatial affordances. Whilst temptingly trivial to plot these over the entire validation set or work out a cumulative sum, the reality is that this would obfuscate and hide any problematic points such as cases where the mean distance between points is significantly non-zero, or 0.

4.4 Vector Location Analysis

A quirky design feature for monitoring training with additional loss terms is multiple items per modality that are meant to converge. The underlying assumption that has preceded this work is that captions generated for the same image have largely the same semantics. It therefore seems prudent to assume that these will converge to each other during training when everything else should diverge.

It is therefore important to check during training that captions from the same text item in fact converge on each other.

Figure 4.10 shows how a single data point moves each epoch. The initial epochs are given a slight negative skew, but the distribution has insignificant deviations between epochs. This suggests that the embedding space is well defined by this method in particular and training is stable within it. An observer familiar with the usual domain of cosine similarity will note that the range here is exceptional for being $(-4, 4)$, these values have yet to be normalised. Normalising these values ensures (as previously stated) that the hypothenuse to the mean is of size 1. This means that the final, normalised version of this graph will have a range smaller than (-1,1), which could only be achieved with multiple 1-hot encodings which would massively diminish the expressability of the network. Most of the values will be within the range $(\frac{1}{\sqrt{512}}, -\frac{1}{\sqrt{512}})$ (approximately < 0.05). This highlights another flaw with einsum approximations, doing the n-way dot product: that multiplying such small numbers results in something inexpressable in low precision, highlighting another limit to that approach.

The graph in Figure 4.11 shows how the embedding of a single data point moves in a Cartesian space during training. To calculate the delta, unnormalised vectors are used. If movements were random and epochs were unstable, the maximum delta would be $\pm$ the range of the embeddings, representing a full swing along an axis. Most of the values are moving ± 4, suggesting that this represents the movement of the peak values rather than random.

A key observation from Figure 4.11 is that during training, the shape, style, and magnitude of the deltas change. This is not a linear correction: In the initial epochs, most changes occur in a range of magnitudes up to 2. Considering the graph produced in Figure 4.10, the overall distribution is the same, suggesting that the Cartesian coordinates are changing, or more accurately, that this individual point is rotating around the Cartesian/polar space. This is a positive indicator that the underlying theory behind having n dimensions and improved scaling is correct: This movement is violent and drastic between epochs, which reinforces the implied scaling that governs this machine learning approach.

The 3rd and 4th epochs are fascinating; the plotted distributions have less deviation from a Gaussian distribution, indicating very subtle changes in the

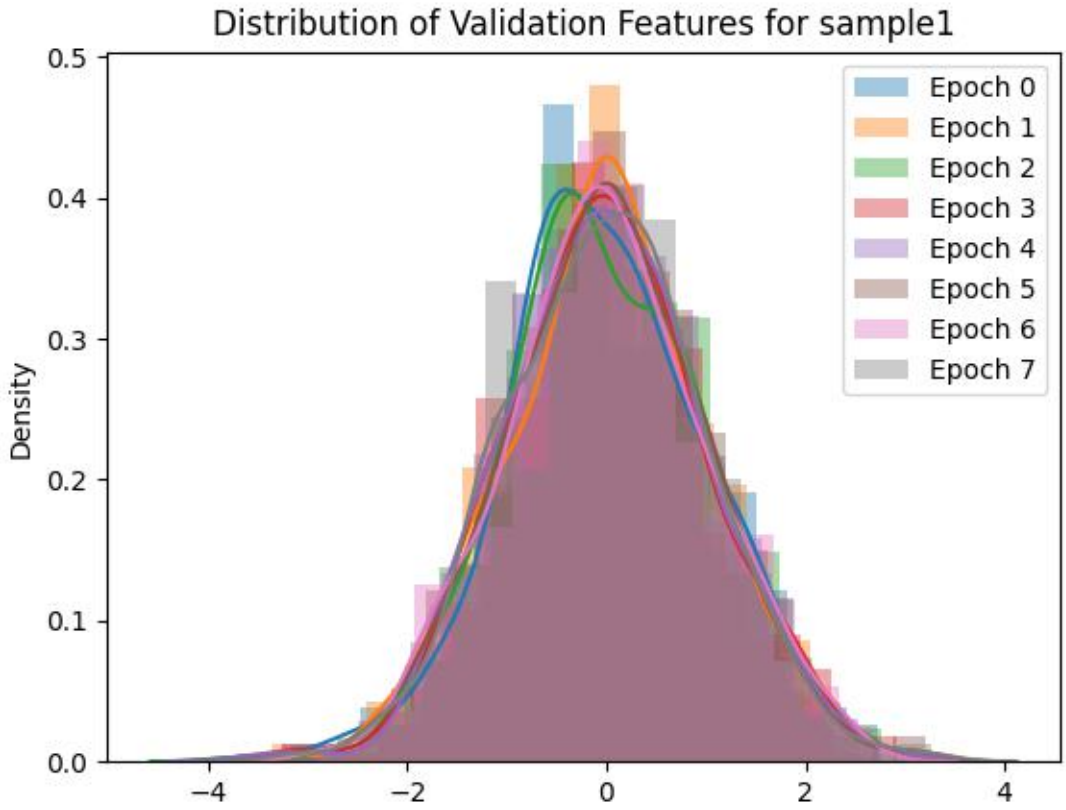


Figure 4.10: Plotting the pre-norm values of a vector of a random datapoint throughout training shows the stability throughout training and the comparative movement of all parts of the vector through training. As every colour is visible on this graph somewhere (assuming this generalised), there are statistical methods that would allow us to predict the epoch by a given vector, indicating the limited data is not sufficient for stable convergence, but that the model is accurately learning the contrastive distribution around the space

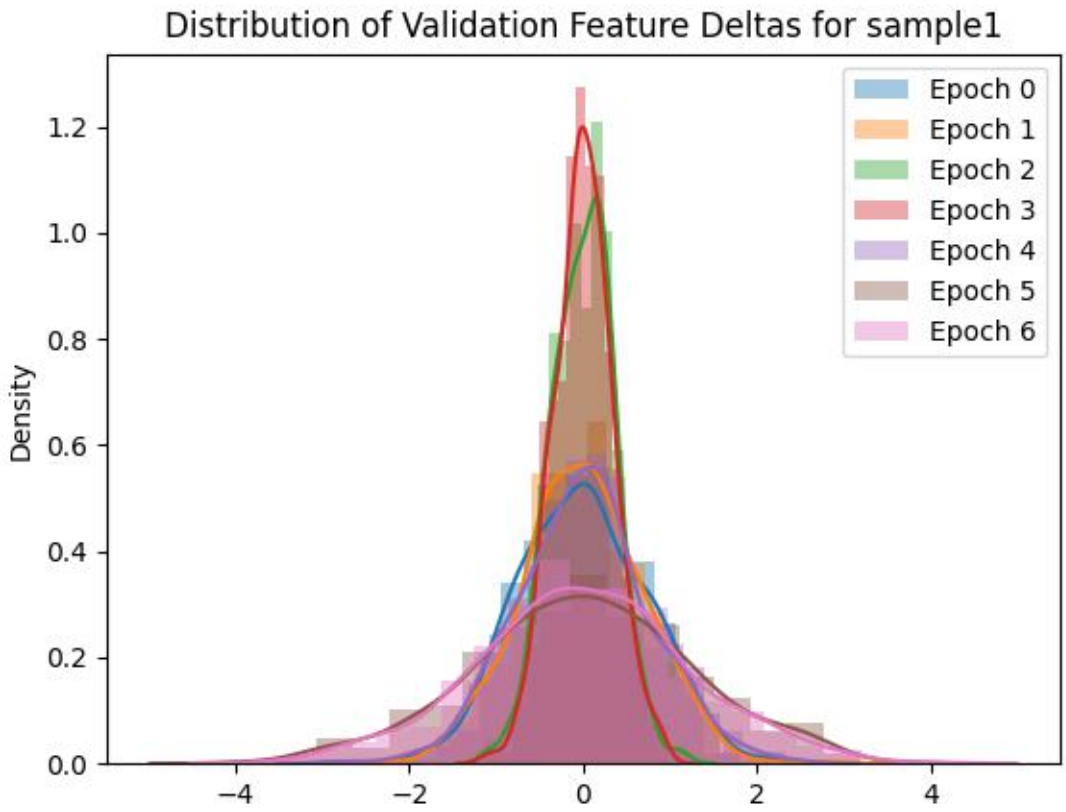


Figure 4.11: A graph showing the change between epochs of a single embedding

embeddings (despite still having the same overall distribution). It is important to consider that these are still unnormalized, suggesting that these changes have a smaller effect on the overall placement of this point relative to other clusters of points in this space.

If training had finished after this small number of epochs, against the initial hypothesis, it would be reasonable to conclude that the achieved embedding space was stable and the model had stopped moving this particular point. However, the final epochs having much bigger variance points indicates a suboptimal solution and has a few explanations:

- Firstly, a single point out of 6 is tracked: the others may be moving more relative to other locations, and our comparatively low batch size results in this set of 6 points having only been compared to 24 others at this moment in training. The result being that this is still comparable to the first step of the base method.

However, this method would scale in a way that is not a relevant time concern.

- Secondly, the cluster of 6 points is moving to a new location: its possible within the space that clusters of embeddings are very volatile, and pre-normalisation, it can take just a single large change to radically alter where a cluster lies in the space. This could indicate that the model is learning how second (and higher) orders of knowledge work, which is a key function of this work.
- It is entirely possible that this training methodology has a learning rate that is too high and becomes unstable near the perfect case. However, Figure 4.10 suggests that as the embedding overall distribution is incredibly stable, this is unlikely to be true.

The graph in Figure 4.10 is strengthened by seeing that the aggregate across a validation batch is largely the same as seen in Figure 4.14. This shows that this is not a unique case for a single set of values but that each value tracks this perfect case across all epochs.

The further consideration of Figure 4.13 shows that the deltas across all distributions follow almost identical distributions, indicating that the total movement is consistent. The aggregate plotted in this graph encapsulates other graphs such as the one in Figure 4.11. This means that the variance in Figure 4.11 is equally offset by other samples having roughly equal and opposite sums. This supports the hypothesis that the movements are unique to smaller clusters of embeddings relocating and having small finite adjustments, and that this delta happens in a way that when normalised represents a rotational movement (otherwise causing a different distribution of values).

4.4.1 Similarity Tracking

Denoting F as the set of features for a batch such that F has a shape of [B,D], the graph should show convergence over

$$\frac{F@F^t}{B^2} - \frac{1}{B}$$

. The term $\frac{1}{B}$ removes the comparison of the elements to themselves. From Chapter 3, it is clear that the similarity of vectors is not 1 due to the limitations of computational accuracy. The operation to calculate the exact value is unnecessary and represents an additional computational cost compared to a fixed constant. Using this value is more transparent to the reader, and allows the graph axis to maintain meaningful limits. The value is plotted throughout the training to show that the annotations of each item are converging.

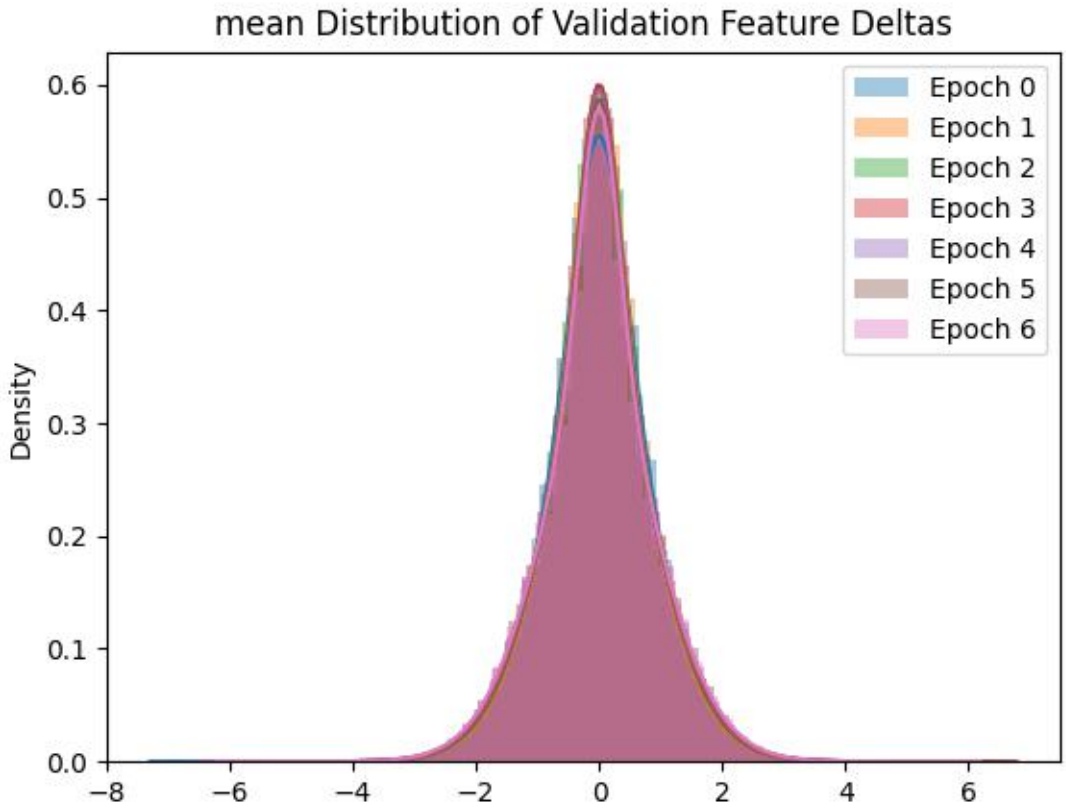


Figure 4.12: This figure shows how the movement of particular encoded points occurs between epochs. It is typically gaussian, emphasising the movement of embeddings is primarily centred around 0, with the occassional extreme value, which is localized to a single feature.

The same score is used, but to measure the first item annotation from each item. By the same metric, the expectation is to see them diverge further away across training.

Understanding the comparative rates at which samples converge and diverge during training is a key insight into how similarity metrics work.

Figure 4.15 shows the $\bar{F}$ produced by the above formula, using the first vector of every batch of captions. A conscious experiment design decision was made to illustrate this using the textual domain features rather than visual ones because, for some experiments with a pre-trained model, the visual features may lead to a misleading result.

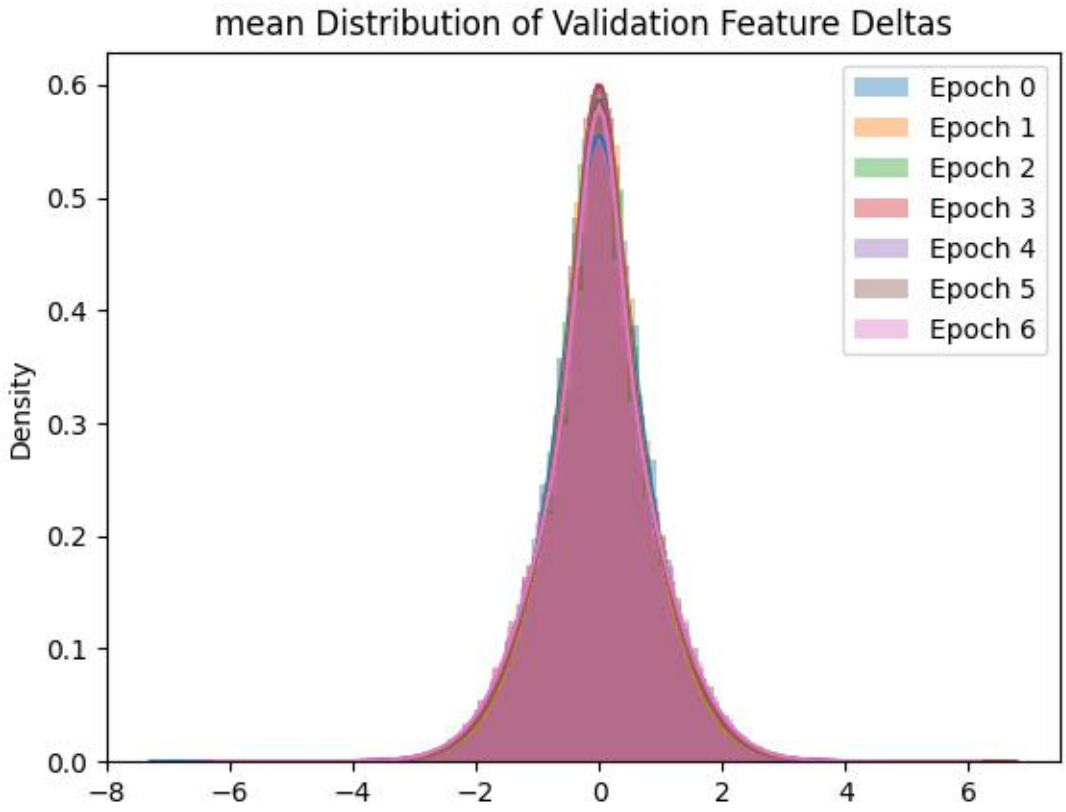


Figure 4.13: The mean plot of all movement deltas across all batches, showing a distinct similarity to Figure 4.11, showing that it is consistent with other items in batch. This graph shows the movement having a higher variance, similar to Figure 4.11, for early epochs.

Figure 4.15 shows a gradual decay in similarity throughout the epochs, indicating that most unrelated batches are moving away from each other throughout the epochs. This shows that training is constantly improving across this number of epochs and results in a cosine similarity near 0, which is synonymous with no correlation between vectors. This characteristic is actually a contributory statistic to other training metrics, but serves as a practical illustration between batches to show that the method holds intra-batch as well as per each step. Figure 4.15 is also instrumental in showing the limitation of even the introduced scaling rules: showing that after several epochs, multiple days on a P100, training is still very far from converging. A P100 is orders of magnitude slower than even the original training GPUs over 4

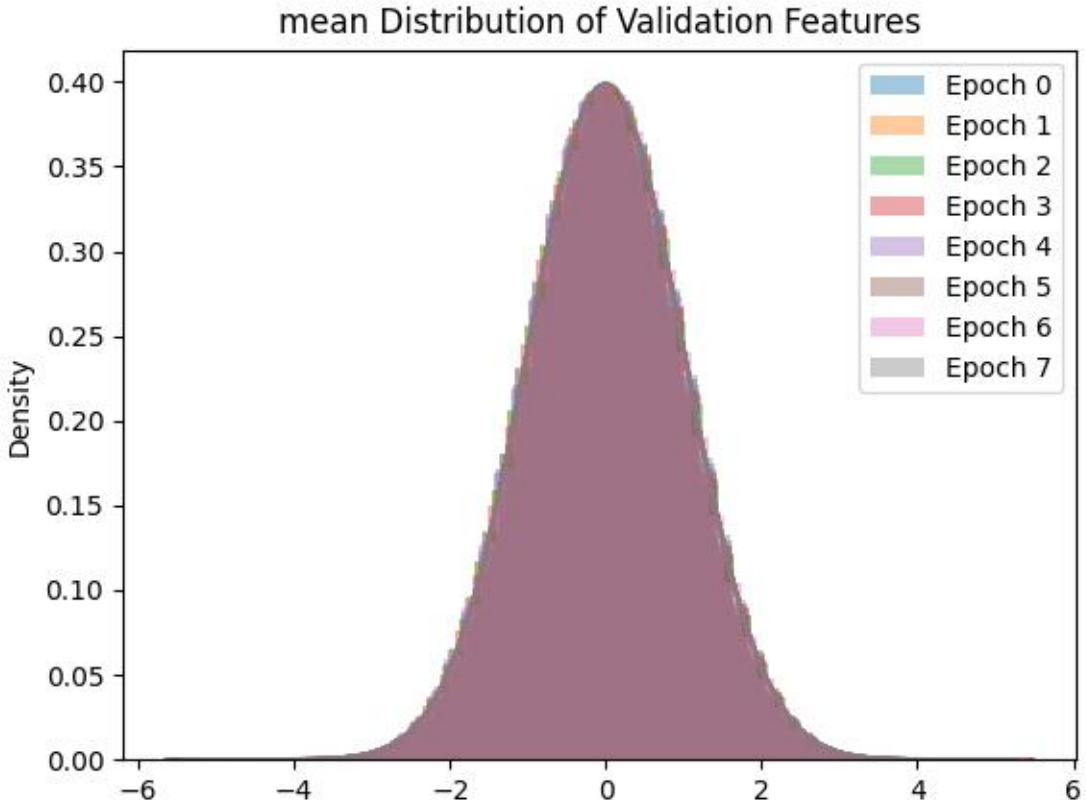


Figure 4.14: Monitoring the distributions of all embedding features, shows that although the embeddings are moving around latent space, they maintain the same value distribution.

years earlier, yet is still beyond academic provision for many. This should serve as a practical warning that even with years of optimization, better support is needed! For the purposes of analysing runs, these graphs provide very intrinsic, tangible, and understandable tracking of vectors. They represent a modest overhead of memory to store batch vectors across epochs, so have had to be largely scaled back from full deployment saturation due to the limits of available hardware (a running theme!). But enough runs have been achieved that these demonstrate a workable sample to indicate hyperparameter configurations that produce runs where the graphs generated are not congruent with the ones demonstrated.

Compared to Figure 4.15, Figure 4.16 compares the average similarity between items in the same batch, which conceptually should all share near-identical embed-

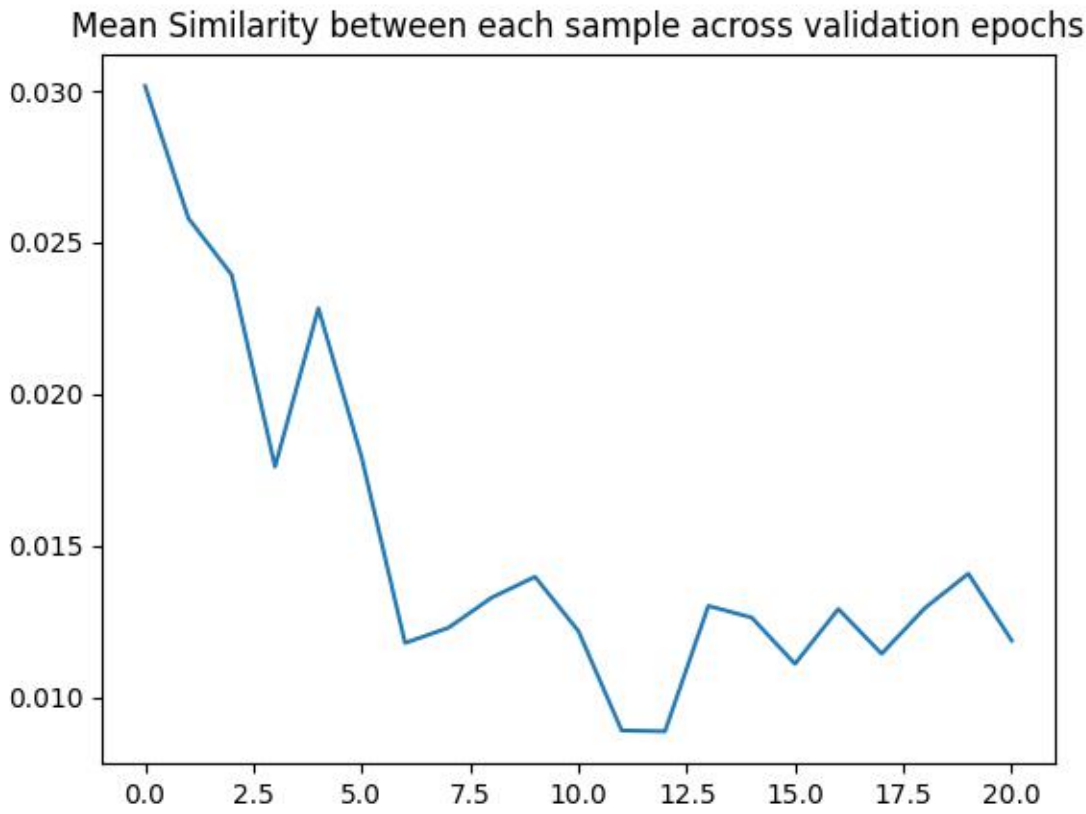


Figure 4.15: Similarity measure between different batches

dings. It is reasonable to expect that the noise created by different annotators can be mapped onto a Gaussian distribution centred around a mean point, but how this abstraction scales to 512-space is not always clear. As a conceptual model, the variance associated is nonzero, meaning the similarity within this cluster of points is also anticipated to be small. An underlying assumption of the original CLIP model manifesting multiple orders of knowledge in a method consistent with other theoretical models is that the clusters of points are smaller than the distance between clusters. The comparison between Figure 4.16 and 4.15 is a very strong indicator that the assumption that the cluster size is smaller than the distance between clusters is experimentally correct. When the assumption is made that a semantic cluster follows a normal distribution, an implied property is that a single value taken from that distribution is the best estimator of the mean of the distribution. Therefore, it can be asserted that the graph in 4.15 generated by taking the first caption from

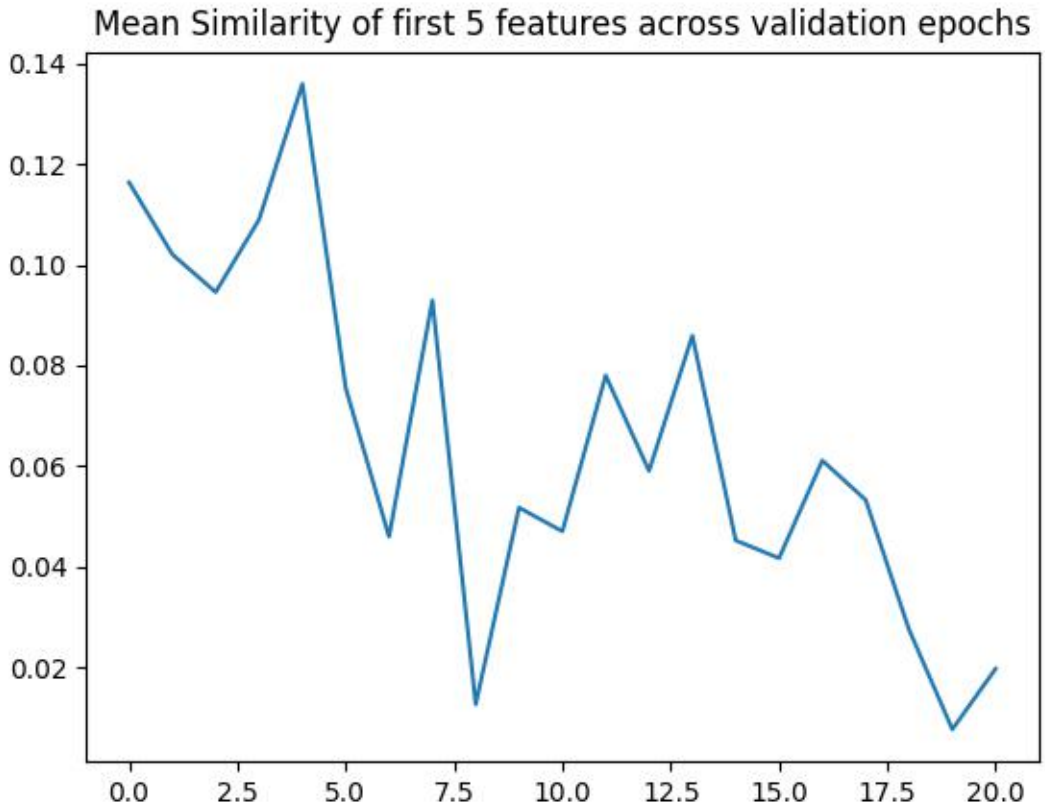


Figure 4.16: Similarity measure in-batch similarity during validation

each annotation is a very strong approximator of the graph for comparing the cosine similarity for the central point of each cluster. If the clusters were closer to each other than the points within each cluster, the values of cosine similarity would be large, because the angle at the origin would be smaller. The training loss would also be unlikely to converge because there would be significant overlap between neighbouring clusters. It can therefore be asserted that the theoretical model is true. However, it must be noted that the matrix used to determine $\bar{F}$ may have high variance, and some vectors may be very close together: therefore, F is calculated throughout the validation epoch, ensuring that every cluster is counted equally and that the matrix is sufficiently large so that a one-hot enumeration of samples could not outperform the resultant graph to minimise cosine similarity.

Figure 4.16 shows a gradual decay in similarity during training. Assuming this is linear (which there is no reason to) would suggest that the intersection would be

around 20 epochs into training. However, the loss is largely at the same proportional rate as the comparative similarities graph between batches shown in Figure 4.15. The proportional rates being approximately the same is a good indicator that the geometries of the embedding space are being preserved.

4.5 Token Meta Analysis

Introduced in the first chapter was the concept of orders of knowledge, which manifests in the geometry of embeddings. The geometric topology of the generated embeddings was key to capturing holistic representations. However, a key constraint of this work is the scale limitation. The notion of this is that there is an objectively correct arrangement that a perfect training will reach. The testing of hyperparameters can be considered distortions on this perfect case, but should not affect the entire distribution of points.

However, given the significant quantity of tests, even with limited movement, a significant amount can be learnt from aggregated training. Given the distribution of image vectors, it is reasonable to assume that the values comprising the vectors are on a normal distribution. The tokens are monitored during training. The embeddings are initialized evenly around an embedding space, so the movement of the tokens should reflect the visual significance in a well-trained model. The machinations of attention mechanisms would infer that the farther the embedding, the more semantically charged the token. This is reinforced by considering context and semantic embedding, where orders of knowledge result in natural clusters. The outer and innermost (relative to origin using Cartesian distance) part of each cluster would reflect semantically unique words.

4.5.0.1 Hypothesis

The expectation is therefore that tokens furthest away are either unique within individual captions and therefore good identifiers, and the innermost are likely to be meaningless against visual grounding.

4.5.0.2 Methodology for Comparing Tokens

With additional parameters and trials, the token lists change size arbitrarily based on dataset, sampling method and how many inputs. This makes the intersection of sets, which should be distinct, in a stable environment a good potential metric for stability. Even though models are non-deterministic, defining knowledge as a series of parallels between points creates natural formations of clusters in an embedding space.

Dimensions	2	3	3.5	4	6
Instability Score σ	0.64348	0.054274	0.42334	0.32875	0.011394
Runs N	127	1233	1015	1298	1152

Table 4.2: This Table shows the Stability scores between a key variable like number of dimensions, which significantly changes the experiment set up.

Using a measure of the outer set of close and far tokens divided by the inner set, the correlation of these sets within training runs across a sweep will be examined.

To test this, each training dimension will be rated with a stability factor. Given the set of all far tokens, temporarily denoted as ψ and close tokens, ω, the stability σ in experiments N can be written using set notation to convey the intersection and union of these sets as:

$$N\sigma = \frac{|\psi \Delta \omega|}{|\psi \cap \omega|}$$

Using this factor, the breadth of learning that occurs within a sweep can be measured by assessing the agreement score of the models trained within that particular parameter.

4.5.0.3 Discussion

Table 4.2 demonstrates the different scores that correlate with each dimension. Dimensions of size 3 and 6 appear to be the most unstable, which partially reflects that both experimental setups have highly irregular sweeps. As can be seen in the code base, the tests with 3 dimensions predominantly had 2 teacher models, and could therefore use a relatively high initial learning rate, unlike the other tests. This metric is can therefore be said to be very sensitive to inconsistent data orders, where a model may marginally overfit or be affected by a relatively small sample size yielded in differing orders. Future work may investigate whether a lower learning rate smooths this metric, and therefore whether σ is a good indicator of optimum learning rate. The other significant outlier in the table is $D = 6$ where the majority of runs vary by how they calculate similarity, which is a significant factor in where embeddings are placed, and therefore the Stability score being low is to be expected, especially given the use of methods that have discussed flaws such as Einsum approximation.

4.5.0.4 Token Set Exploration

Although these sets are gathered across the entire set of experiments, notably across all scales, there were no tokens that appeared in the set of closest tokens, which also appeared in the outermost tokens. This is a significant finding because it

shows that the model methodology is working consistentlyand provides an additional unsupervised methodology to observe poor training paradigms.

Using these aggregated token lists provides a further metric for training by looking for statistical overlap per run.

The congruence of token lists with vision-based evaluation metrics. Define the token score T as the product of the intersection sizes of the closest and the farthest sets.

A core limitation to the exploration of this method is that analysis has occurred on logged data after a significant portion of the experimentation has occurred. To ensure verbosity and significance, it should be repeated, and a significant improvement would be to use either a fixed fraction of tokens or a cutoff for what comprises near- and far-clusters in 512-space. Such improvements create a more meaningful comparison of the learnt space and improve the fidelity of such a score to measure the congruence of training. There are arguments to be made that such approaches could be applied to using a comparison of embedding weights, and monitoring their movements, but a large body of work would need to be implemented to ground this against the semantics of individual tokens and dataset entries, there are also problems where the embedding space is not topologically constrained by outputs resulting in false negatives. However, applying this set theory approach produces a granular, but readily interpretable metric, leveraged on cartesian distance that allows a topological freedom of embeddings.

Using 2 dimensional loss, the baseline runs, across different parameter sets, the following graphs are generated:

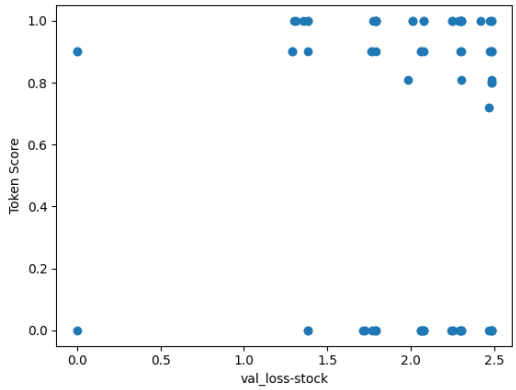


Figure 4.17: Correlation between TokenScore and Validation Loss in 2 dimentions

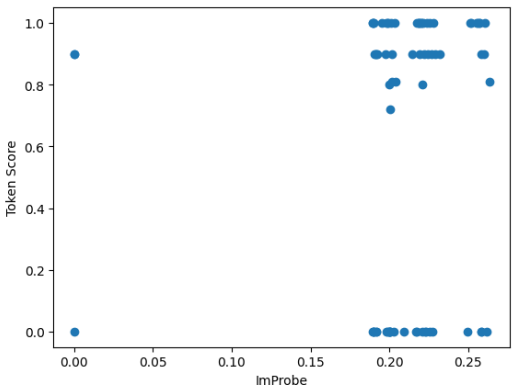


Figure 4.18: A Plot of Token score against image probes in 2 dimensions

Notable from the above figures is that ignoring the outliers (caused primarily where NaNs result from poor logging) shows that the Token score correlates strongly to the

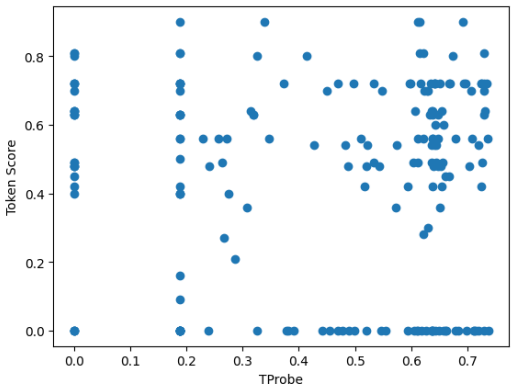


Figure 4.19: A Plot of Token score against text probes in 6 dimensions

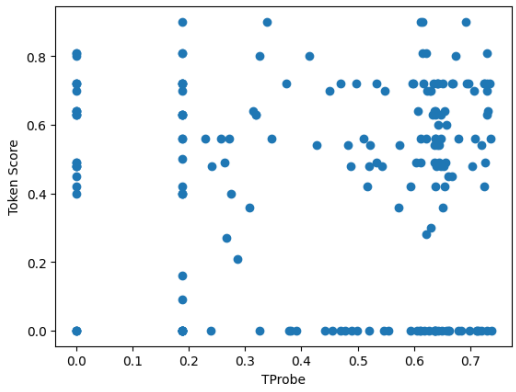


Figure 4.20: A Plot of Token score against image probes in 6 dimensions

other key metrics, is domain and corpus irrelevant, and aligns to both Text and image probes and can be calculated through training. In this work implementation, the tokens were decoded to provide a direct human-interpretable output to spot further patterns; the process does not require decoding.

4.5.1 Token Analysis Summary

The plots in Figures 4.17-4.20 show how Token analysis correlates to other metrics. Ignoring outliers where Text Probes or Token score is zero, there's a correlation to training heuristics like linear probes, but it is very weak. Compared to prior work's methodology, there is no meaningful correlation. This suggests that this analysis is not a meaningful metric compared to the prior similarity score.

4.6 Chapter Summary

Concluding this chapter's work investigating **RO.2**, various evaluation methods have been presented which innovate on prior work for evaluating the performance of autoencoders whilst adding minimally to computational overhead. CKA is presented as a useful metric for model evaluation, enabling comparison of layers, both to neighbouring and distant intermediate layers, showing different regions of a model and how they interact and compare. CKA has been reengineered in this work to be several times more efficient as a major contribution in reducing footprint. A drastic improvement to the existing implementation of CKA has been offered and has been shown to be correlated with good training cases.

Building upon the methods presented in prior and subsequent work, the training

methodology has been explored to show how tracking correlated data points can provide significant insight to model performance. Logging the latent embeddings demonstrates that the methodologies presented in this work scale, and can be applied across modalities, which is significant for the upscaling of this work as it impacts industry. The methods discussed build upon the principles of the underlying models and the pre-requisite assumptions that similar inputs should result in similar outputs. Having stated these training heuristics, and demonstrated they are still consistent in $n > 3$ training, a method is available for assessing the efficacy of models proposed in Chapter 5.

A novel inter-training scheme has also been shown, which while limited in this context with weak supervision, is a good candidate approach for training heuristics in rapid prototyping and black-box domains, especially given the rate of growth of the AI industry, where company strategy may call for lavish experimentation because the cost of delays is less than that of excess computation. Alongside these novel methods, they have been shown to have good fidelity to model performance in the training paradigms used in this work, and will be used going forward to evaluate understanding in such systems.

Chapter 5

Methods of n-dimensional loss

In previous chapters, Contrastive loss has been explored, showing its significance at higher dimensions, and the capabilities for rapid teacher-student training of an encoder, especially in a poorly resourced domain where a well-trained teacher model is invaluable for instilling general knowledge.

As previously discussed, a core component of the CLIP model is the comparative locations across vectors. This is performed as a high-dimensional cosine similarity. Cosine similarity between two vectors is performed by normalizing the values across the vectors to be in the range $-1, 1$, with a hypotenuse length 1, which are then multiplied element-wise and averaged. This provides a value for the similarity between -1 for opposites and 1 for a perfect copy, respectively.

In this chapter, the research presented achieves the following:

- Several methods are presented and compared for measuring similarity of n vectors
- Efficient computation of loss is explored for more dimensions
- Model performance is compared using each different method.

These will be achieved by firstly demonstrating the scaling laws present, and exploring the hypothesis that there are economies of scale to be gained. This task is broken up into smaller steps: addressing the mathematics of $n > 2$ as a data structure; finding the practical upper limit for n in a machine learning pipeline and finally profiling the performance of similarity metrics.

Using contrastive training for supervision so far has been promising. It can be fairly assumed that with greater scale and the multimodal capabilities that come with Scale as in the case of CLIP, this will be a low-cost way of training a transformer encoder to a low-resource domain using the teacher models to infuse a visual grounding.

5.1 Research Objective

So far, the discussion has highlighted the many advantages available to models that use weak and pairwise supervision to learn the relevant associations. So far, it has been shown to be very effective to use tertiary encoders to increase the potency of limited data and to utilise better scaling ratios. In this chapter, **RO.1** will be continued and **RO.3** will be explored in the form of **RQ.2**. The efficacy of the contrastive method is often attributed to multi-modal encoders co-learning useful representations. The subsequent research question would be how to adapt the CLIP methodology to see how multiple encoders learn simultaneously across multiple modalities. How does the efficacy of training improve (data permitting) with multiple encoders learning simultaneously, and what are the limits in numbers of dimensions? This chapter investigates the scaling laws that govern the efficacy of CLIP training, and investigates whether that relation extends beyond 2 encoders, and therefore whether the efficacy is related to the number of encoders, or a quantity of data. Continuing from the conclusion of the previous chapters in answering **RQ.2**; this chapter will answer whether there is an upper bound to this scaling between different methods.

5.2 Base Method - CE loss pairs

The most efficient way to bridge modalities with additional dimensions is to iterate over pairs of dimensions. We consider that the dimensions give us a tensor stack of shape $[n, B, F]$. We want to multiply these matrices $[B, F]@[F, B] = [B, B]$. We want to scale this up to be a batched multiplication between $[n, B, F][n, B, F]$ to get a sum over n.

To perform a baseline of this method in an academic setting, the methodology is optimised, to perform a matrices multiplication similar to an einsum with $[n, b, f].[n, b, f] -> [n, n, b, b]$. The result is reshaped to get a stack of $[B, B, n \times n]$, which can be used to perform traditional CE loss.

Performing cosine similarity with multiple terms is nontrivial, as explored in the published paper which lays an accepted foundation for this chapter. The operator for Batch Matrix multiplication only accepts two operands. In this section, the aim is to explore effective scaling with additional streams of information. During iterated pairwise training, the outputs, O, where O_i is the i^{th} output if it can be paired. There are therefore $\frac{i^2-i}{2}$ or $iC2$ operations per step, which removes any advantage of scaling, as the number of datapoints in the dataset increases.

Alternatively, some work exists around the base methods to aggregate batches, creating a grid of shape $NB \times NB$. This hits the same limitations as CLIP, in scaling poorly with Batch size, splitting the calculation in a similar way to attention heads would split the QKV formulation in transformers. It also involves a significant amount

of calculation pre-reduction, though it could readily be applied with techniques like gradient caching. We also hit scaling issues as N or B increases, especially as the batches that begin and end within the logit grid must be monitored in order to aggregate at the end. The goal of finding an algorithm that uses additional dimensions is the trade-off in logits.

CLIP training took 512 GPUs for 19 days. Although the synthetic batch size was some 2048, it is reasonable to look at implementations that suggest the gradient calculation used batch size was 64, subsequently aggregated between devices. This means that the logit available for the loss computations was 64^2 or 4096. If we take a conservative estimate of our function, assuming that it will take a batch size of 8 across 6 dimensions, this gives 262144 logits, a factor of 64 $\times$ better. This means that we might expect this algorithm to be competitive with just 8 GPUs in 19 days. If we can increase batch size to 10, resulting in 10^6 logits, the runtime can be effectively quartered to 2 GPUs in 19 days. If a batch size of 12 is achievable, the same logic dictates that only 13.3 GPU hours are required.

To formalise this criteria, the following checklist outlines the goals explored in this chapter:

- Returns a measure of vector similarity for N terms, where $N > 2$
- Has no false minima that are independent of inputs.
- Can be calculable within the confines of a single GPU
- Competitive with CLIP after 16 hours.
- Unaffected by the polarity of vectors.

5.3 Scaling Laws

There are many variations and rationales available for high-dimensional loss, with different merits and behaviors. However, the rationale remains the same. If the model proposed training is abstracted and compared to CLIP, with the same number of parameters present, the training limit on a single device is equivalent to the maximum value of B gradients for each encoder. The efficacy of each gradient is a function of B because the ratio of in-batch negatives can be expressed as $1 : B$, it is unclear how this may scale with training, whether there will be diminishing returns as B increases, or whether B will improve training by orders of n. However, other factors like memory limits exist: for training n data inputs, B must be n times smaller for each encoder, but the volume of in-batch negatives scales as $\frac{B}{n}^n$. Substituting in the ratio of batch negatives into the formula as $\frac{B^{n-1}}{n^n}$. In summary, this is the ratio of in-batch negatives

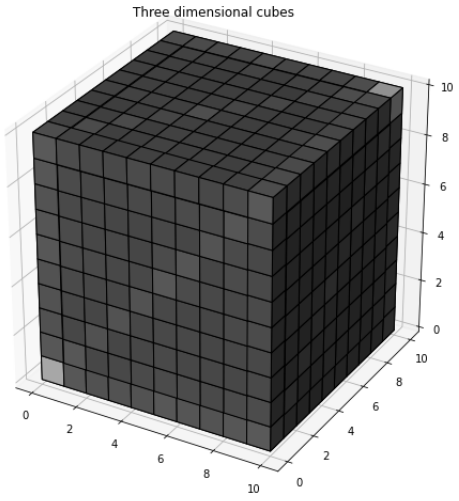


Figure 5.1: A plot of logits in $n = 3$ dimensions, showing how in $n > 2$ dimensions, there are more than 2 distinct values in a best case

over the trade-off of a limited batch size. Given that n is fixed for a given architecture and training style, the effect of B is significantly greater! To demonstrate the reason this is significant, in the following, the dimensions $n = 6$ are used, where $B = 8$ is achievable with 16GB VRAM. On the same device with $n = 2$, $B = 48$. The in-batch negative ratio of 1:32768 vastly dwarfs 1:48, which would require 683 steps over the same data (assuming batch shuffling) to compare. Doing equivalent ratio training with $n = 2$ is simply not achievable with anything short of 100's GPUs, which still takes significantly longer due to the requirements of gradient and batch aggregation across devices.

5.4 Scaling Structures to $n > 2$

In this chapter, the working hypothesis is that the additional affordances of having more than 2 dimensions to contrastive training improve performance. To establish an empirical test, and explore the effects that this approach has on training, these extra affordances must be defined and quantified.

5.4.1 Defining the Regions of $n > 2$

Firstly, one of the core problems of the increasing number of logits is the arrangement. A simple calculation of similarity values now produces a shape as seen in Figure 5.1.

n	1	2	3	4	45	5	6	7	8	9	10	11
Number of Regions	1	2	3	5	7	11	15	22	30	42	56	77

Table 5.1: A table showing how the number of distinct regions scales with n

In Figure 5.1, the important aspects is the diagonal where the coordinates in the cube exist on the plane $x == y == z$, all white in the plot, showing the desired values there are at the maximum similarity value (1). This corresponds to the desire for batched outputs to relate to their corresponding batch outputs from other encoders.

Notably within Figure 5.1, the slightly lighter diagonals reflect a quirk of 2+ dimensions where there are points with coordinates that are neither the same nor entirely unique. This means that parts of our n-dimensional cube will naturally be higher than others. These can either be ignored, supposing that the unique indexes need to be far enough removed to maintain a poor overall similarity value. However, the alternative approach for a more stable gradient is to model this pattern into the loss metric. In the example of Figure 5.1, it is clear that there are 3 distinct regions, each with different behaviours.

In cases where $n > 3$, a way of selecting behaviours by region is needed. Given that in the region that a particular coordinate belongs to is related to the relation between coordinates, a formula can be engineered.

When $n = 2$, or $n = 3$, the number of distinct regions is linear: all coordinates are the same, all are different, or (as in the case of $n = 3$) some may be the same. Beyond $n > 3$, the number of regions becomes non-linear, which are shown in Table 5.1.

Tracking all the combinations requires the construction of a cube, c, such that the value at location can be expressed as $c_{ijk} = (i, j, k)$ where $0 < i, j, k < B$ for $n = 3$, using more coordinates as needed. In this cube, this can be reduced to give c' the counts of each unique value in each cell such that the total is n for each position in c. However, to make the distributions unique, the value placed on each region is as follows:

$$\text{MaskValue} = \sum_{i=0}^{B} c_i'^m \tag{5.1}$$

Calculated over the whole cube, this gives a cube of where any locations that share the same value ought to have the same behavior during training.

Figure 5.2 shows how more regions may appear when projected onto a 2D graph. In Figure 5.1 these can be seen as the three different shades, but in larger dimensions, more distinct regions are visible. The flat projection of this graph is generated with

```
#Generated with:
logits.flatten().unflatten(0,(B*B,B*B)
```

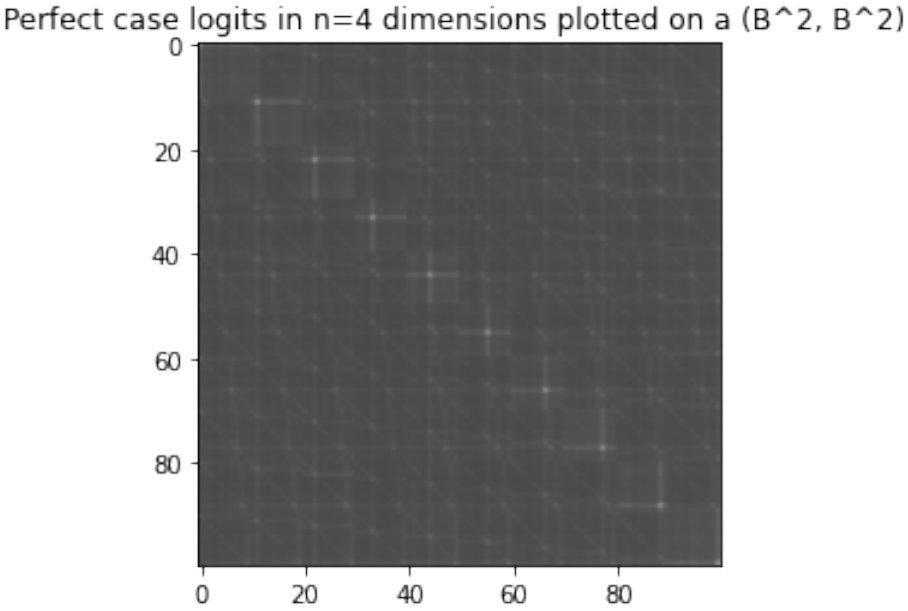


Figure 5.2: A demonstration of the 5 distinct patterns the perfect case for $n = 4$

Initially, when masks are calculated, an identifier, m, is used for each region. m is given the value of the sum of the counts of each coordinate squared. For a small number of dimensions ($n < 6$), this is fine and results in unique values, but at $n = 6$, there are conflicts where multiple sets have the same mask.

The initial experiments used $m = 2$ until unusual results were observed, due to a set of coordinates (MaskValue = 12) having the same mask value as a different configuration. For $m = 2$, 3 pairs of matching coordinates $(0, 0, 1, 1, 2, 2)$ have the same mask as a set of 3 matching coordinates $(0, 0, 0, 1, 2, 3)$. There is also a conflict at $m = 18$ with a set of 4 matching or 2 sets of 3, such as sets $(0, 0, 0, 0, 1, 2)$ and $(0, 0, 0, 1, 1, 1)$. When considering whether this would impact the experiments and separation of behaviors, it would not make fair subsequent tests. Manually reassigning these masks is possible, but detrimental to deploy during training due to the branching introduced. Instead, it is computationally simpler to increase the exponent. The first value for the exponent across all tests of n is $m = 6$. This value was settled on as demonstrably stable for the scope of this work, though it is expected for future works that further investigation is warranted in larger scales.

By masking the loss for select regions, the loss component is demonstrably affected by the relative scale of each region. Therefore, each region is scaled by relative density when comparing multiple regions.

When properly optimized, the mask can be calculated very efficiently using the following code:

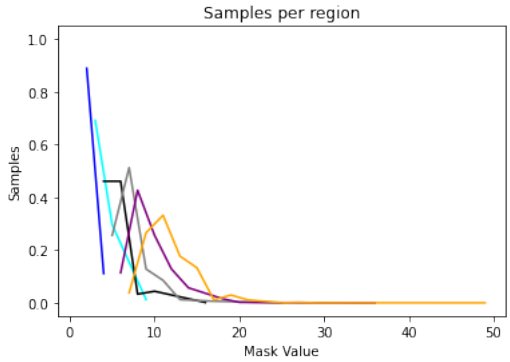


Figure 5.3: A figure showing the proportions of a logit cube attributed to each mask value as B increases for $n = 6$

```
# Initialize batch size and N
B = self.hparams.batch_size
N = 6

# Create a tensor Views with values based on batch size B
Views = diagonal matrix of size (N, N) where each diagonal element is B-1, then add 1

# Compute bincounts2 using one-hot encoding and summing the results
Bincounts = sum of all one-hot encoded tensors generated for each element in Views

# Calculate the Lossmasks as the sum of bincounts2 raised to the power of 4 along the last ↩
    dimension
Lossmasks = sum of Bincounts^4 along last dimension

# Create the masks by flattening Lossmasks and removing duplicate values
Masks = unique values from the flattened Lossmasks tensor

# Ensure that the label shape matches the Lossmasks shape
assert label shape == Lossmasks shape
```

5.4.2 Exploring How Batch Size Effects the Number of Regions

The discussion, and calculations that form Table 5.1 suppose that the batch size is a substantial number. There are some cases where this is not true, such as where $B \leq n$. As an intuitive edge case, when $B = 1$, the number of regions will always be 1, irrespective of n. In this part, the relation will be investigated, as the size of each region is likely to have a significant factor in determining gradient contribution in later experiments.

Therefore, Figures 5.3, 5.4, 5.5, 5.6 are plotted to demonstrate how the relation between regions, batch size, number of dimensions, and the proportion of the logits that each region accounts for relates.

Figure 5.3 shows how the proportion of a given cube is attributed to each mask, plotted for different batch sizes. From this graph, it is important to note that the proportions become more stable with batch size and tend towards larger mask values.

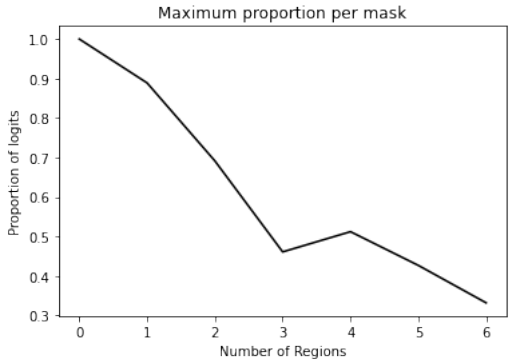


Figure 5.4: A figure plotting the maximum proportion of the cube occupied by a single region as n increases

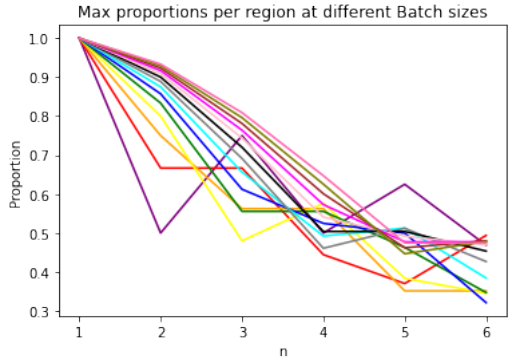


Figure 5.5: This plot shows how the maximum proportion of logits per value of n changes across Batch size,plotted for $B < 14$, stabilizing beyond $B = 2n$

It appears from Figure 5.3 that the maximum proportion of each cube accounted for changes in a manner that is inversely quadratic. To explore this, Figure 5.4 is plotted to show how the maximum proportion scales with the number of regions for each value of n. Figure 5.4 appears to have an outlier at $n = 3$. Given the expectation was to have an inverse quadratic function, the overall linearity of this graph reflects that the scale is too small to find this correlation, or more likely, that the aggregation of samples to a ratio obfuscates the overall number of logits increasing with each step.

From Figure 5.5, it is clear that there is a significant amount of turbulence where batch size is near n. It is interesting that beyond $n = 4$ there is a significant turbulence in the graph. It seems that a similar stability may be present for $n > 4$, should even greater batch sizes be plotted.

Comparing Figures 5.5 and 5.4, it is clear that Batch size has a significant smoothing factor, but is only part of the picture. The conclusion drawn from Figure

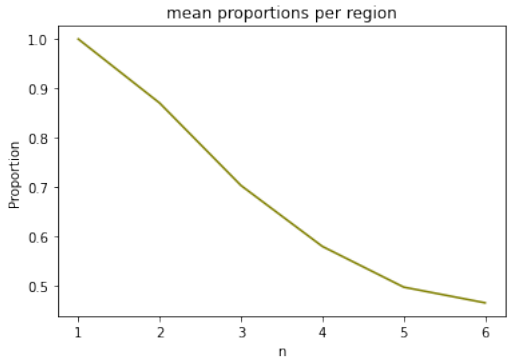


Figure 5.6: A plot showing the mean values of region proportion across logits

5.4 was that the increasing volume obfuscates relations that rely on proportions. To evaluate whether Batch size increase is hiding this because it exponentially increases the volume, the mean proportions in the cube are plotted. This is the aggregated across each value of n as $\sum^{\text{Number of regions}}$ Proportion of Region $\times$ Region Volume

The overall decrease in maximum proportion as batch size and n increase is promising for the hypothesis that increasing n will lead to more effective training. It suggests that the different interactions and differing similarity values balance as training increases, and therefore, are all equally valuable in contributing to gradient over residual attention in the model architecture.

For the further exploration of the **RO.2**, the scaling of batch size, and the composition of the n-dimensional similarity matrix must be considered against how loss is calculated.

5.5 Methods for Similarity in n-dimensional

In previous chapters, little discussion has been dedicated to the comparison of multiple vectors. There are very few established methods for calculating the closeness of 3+ vectors, and whilst cosine similarity has some established modifications for 3 terms, they are computationally expensive.

In this section, various methods are presented, each implemented and tested, based initially on matching the correct gradient range using various approaches, including Cartesian and polar distances, with various modifications for high-dimensional vectors and normalising the values. The methods were proposed on the basis of several desired points for perfect and imperfect cases and on the basis of trying to find a relatively smooth transition between those states. There are many algorithms that needed testing, as can be demonstrated by the above figure showing how the gradient of similarity functions can be radically altered by the number of dimensions.

The loss of a single $[B, B]$ logit matrix using CE loss can be expressed $\frac{\text{Log}(\text{trace}(e([B,B])))}{e([B,B])}$, or more formally,

Given a set of paired samples$(x_1, t_1), (x_1, t_1), \ldots, (x_i, t_i)$, where (x_0, t_0) represents a positive pair and all other pairs (x_i, t_j) where $i \neq j$ are considered negative. For the positive pair, the cosine similarity is given by:

$$S_p = \frac{x_i \cdot t_i}{\sqrt{({x_i}^2)} \times \sqrt{({t_i}^2)}}.$$

For the negative pairs,the cosine similarity is given by:

$$S_n = \frac{x_i \cdot t_j}{\sqrt{({x_i}^2)} \times \sqrt{({t_j}^2)}}.$$

LetS_n denote the similarity score for the negative sample. Then:

$$\text{Loss} = -\alpha \log \left(\frac{e^{S_p}}{\sum_{n=0} e^{S_n}} \right),$$

To differentiate this, it must be rearranged and simplified in this expression:

$$\text{Loss} = -\alpha S_p + \alpha \log \left(\sum_{n=0} e^{S_n} \right)$$

Now, the loss function with respect to S_p in parts as

$$\frac{d}{dS_p}(-\alpha S_p) = -\alpha$$

$$\frac{d}{dS_p} \left(\alpha \log \left(\sum_{n=0} e^{S_n} \right) \right) = \alpha \cdot \frac{1}{\sum_{n=0} e^{S_n}} \cdot \frac{d}{dS_p} \left(\sum_{n=0} e^{S_n} \right)$$

Thus:

$$\frac{d}{dS_p} \left(\alpha \log \left(\sum_{n=0} e^{S_n} \right) \right) = \alpha \cdot \frac{e^{S_p}}{\sum_{n=0} e^{S_n}}$$

Putting it all together, the total derivative of the loss function with respect to S_p is:

$$\frac{d\text{Loss}}{dS_p} = \alpha \left(\frac{e^{S_p}}{\sum_{n=0} e^{S_n}} - 1 \right)$$

This expression shows how the loss function changes with respect to S_p as a ratio of positive and negative samples.

The usual notation has factors like w_c which have been removed compared to the PyTorch documentation, because in the implementation of contrastive training, all the labels are of equal importance.

For scaling beyond $n = 2$ as per prior discussion, the goal at this scale is to not increase memory consumption. The question is whether there is a similarity measure such that our gradient ($\frac{d\text{Loss}}{dS}$ can be computed by parts, or greedily, so as to minimize the need to instantiate the values of a computational graph over a matrix of size B^n.

For brevity, not all methods will be fully explained, as they can be derived from others; instead, they are listed in Table 5.9.

5.5.1 Einsum Approximation

An entirely alternative way to deal with the intricacies of operations between n-batches of high-dimensional vectors is to ignore conflicts. This assumes a low frequency of polarity inconsistencies with cosine similarity in perfect cases where all samples match. The underlying assumption here is that the use of multiple encoders may be pre-trained (for example, by using pre-trained CLIP encoders). It might therefore be reasonable to assume that these representations of inputs will be highly similar. Therefore, concerns about polarity shifts are less likely; however, this will still cause all features to have inconsistencies in their magnitude. Perhaps, it can be solved by scaling the components rather than their products and adding a loss measure as distance to magnitude one.

Pytorch has an operator that accepts any number of matrices and performs multiplication on them by adding a dimension as instructed.

This means that for n tensors of shape B, F, and desiring a cube of shape B^n, the command is

```
inputs=[image_features,
caption_features1,
caption_features2,
caption_features3,
caption_features4,
caption_features5]
logits=torch.einsum("az,bz,cz,dz,ez,fz->abcdef",
*inputs)
```

The Einsum operator has many uses and can perform the vast majority of operations required for ML models. This is poorly optimized for operations with multiple terms; thus we decompose our operation into intermediate steps with fewer operands.

```
loss = self.loss(logs*torch.einsum("abcz,defz->abcdef",torch.einsum("az,bz,cz->abcz",↩
    cache3,cache4,cache5),torch.einsum("az,bz,cz->abcz",cacheim,caption_features1,cache2))↩
    ,labels)
```

Using intermediate values in this fashion is significantly faster and less memory intensive, offering speed improvements of around 23% and 10% less memory usage.

```
function CalculateLogits(I, C, C1, C2, C3, C4):

    # Step 1: Normalize the tensors I, C1, C2, C3, C4
    I = I / norm(I, dimension=-1, keep_dimension=True)
    C1 = C1 / norm(C1, dimension=-1, keep_dimension=True)
    C2 = C2 / norm(C2, dimension=-1, keep_dimension=True)
    C3 = C3 / norm(C3, dimension=-1, keep_dimension=True)
    C4 = C4 / norm(C4, dimension=-1, keep_dimension=True)
    C5 = C5 / norm(C5, dimension=-1, keep_dimension=True)

    # Step 2: Compute squared norms and reshape tensors for broadcasting
    I_squared = square(I)
    C1_squared = square(C1)
    C2_squared = square(C2)
    C3_squared = square(C3)
    C4_squared = square(C4)
    C5_squared = square(C5)

    I_squared_reshaped = reshape(I_squared, [shape_of(I)[0], 1, 1, 1, 1, 1, -1])
    C1_squared_reshaped = reshape(C1_squared, [1, shape_of(C1)[0], 1, 1, 1, 1, -1])
    C2_squared_reshaped = reshape(C2_squared, [1, 1, shape_of(C2)[0], 1, 1, 1, -1])
    C3_squared_reshaped = reshape(C3_squared, [1, 1, 1, shape_of(C3)[0], 1, 1, -1])
    C4_squared_reshaped = reshape(C4_squared, [1, 1, 1, 1, shape_of(C4)[0], 1, -1])
    C5_squared_reshaped = reshape(C5_squared, [1, 1, 1, 1, 1, shape_of(C5)[0], -1])

    # Step 3: Add the reshaped tensors element-wise
    combined_squared = add(I_squared_reshaped, C1_squared_reshaped)
    combined_squared = add(combined_squared, C2_squared_reshaped)
    combined_squared = add(combined_squared, C3_squared_reshaped)
    combined_squared = add(combined_squared, C4_squared_reshaped)
    combined_squared = add(combined_squared, C5_squared_reshaped)

    # Step 4: Take the square root of the combined squared tensor
    sqrt_combined_squared = sqrt(combined_squared)

    # Step 5: Compute the sum of all elements in the result
    result = sum(sqrt_combined_squared)

    # Step 6: Return the result
    return result
```

Figure 5.7: Code showing the approximation of Cosine similarity for 2 dimensions,

5.5.2 Euclidean Distance and Other Methods

We explore using Euclidean distance as a facsimile of Cosine Similarity between normalised vectors. When normalised, each vector x is divided by the norm of x as defined by $norm(x) = \sqrt{\sum x^2}$. When this has happened, most of the values in x are in the range $-0.5 < x < 0.5$, this transformation of values means that the Euclidean distance of the values becomes a good approximator of Cosine Similarity, and Cosine Similarity between x and y may be defined as $x.y$ when Euclidean distance is $\sqrt{(x-y)^2}$. In simple terms, when x and y are considered high-dimensional vectors, this performs the Pythagorean theorem to get the length of a hypotenuse across high-dimensional space.

Instead of treating each additional dimension in x as an additional dimension of the Euclidean distance, we consider the distance defined by x_i, y_i, z_i. This distance can be considered a vector from the norm, of length, $E_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$. In turn, this can be considered as $E^2 = x^2 + y^2 + z^2$, which also scales to additional dimensions.

On a graph, the difference between Euclidean Distance and Cosine similarity between the ranges of $(-1, 1)$ for 512-dimensional vectors is as follows in Figure 5.8.

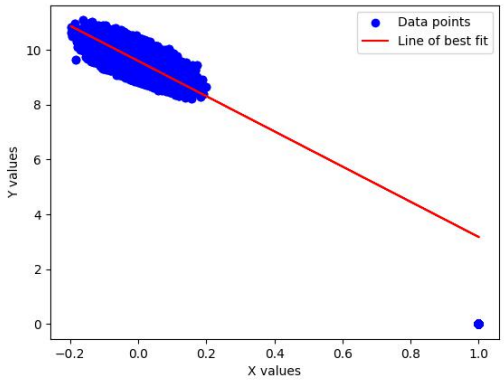


Figure 5.8: Cartesian Distance against Cosine Similarity for Gaussian vectors

Importantly, this shows that even though they can have very similar performance in low dimensions, the outlier in the graph shows how the Cartesian distance is high for any perpendicular vector, but lower when polar opposite.

5.5.2.1 SST Distance

Approaches that treat each feature separately within a feature space can be adversely affected by features with a high range. In contrast, another way to measure distance is to consider the distance to the mean. Basically replacing the similarity value, S with $S = 1 - \text{Variance}$ facilitates the calculation of the variance to be used for loss.

There are several factors to explore, such as the objective distance and the square distance (SST - Total Sum of Squares). The objective distance is simple; When the mean ($\bar{x}$) is calculated, we can let our sum of distances be $\sum |x - \bar{x}|$.

However, when the absolute distance from the mean is minimised, there are cases where undesirable treatment of outliers may occur. Considering the set: $0, 0, 4, 12$ the mean is 4, the sum of absolute distances is 16 but the mean square error is 24. However, there exist worse cases such as $1, 0, 2, 13$ where the mean and absolute distance are unchanged but the mean square error is now 25.

Using this formula of Mean Square error for variance where similarity, (S) is defined as $S = 1 - \text{Variance}$ where we equate variance as the sum of normalised distances to the mean and can be considered as

$$\text{Variance} = \sum_{i=0}^{n} (x_i - \bar{x})^2$$

Notably, This is very similar to the SSG formula and is a factor of n larger than the common formula for Standard Deviation. This is chosen to reflect that the standard

deviation reflects the statistical distribution of single discrete variables, rather than the distribution of vectors. The higher the value of F, our feature space, the larger this distance should be.

Because calculating $\bar{x}$ requires an n sized stack of logits from each encoder projected into a shape B^n, this is prohibitively expensive. However, the equation can be rewritten by expanding brackets.

$$\text{Variance} = \sum_{i=0}^{n}({x_i}^2 + \bar{x}^2 - 2x_i\bar{x})$$

The terms can be grouped into the following.

$$\text{Variance} = \sum_{i=0}^{n}({x_i}^2) + \sum_{i=0}^{n}(\bar{x}^2) - \sum_{i=0}^{n}(2x_i\bar{x})$$

and simplified down to

$$\text{Variance} = \sum_{i=0}^{n}({x_i}^2) + n\bar{x}^2 - \sum_{i=0}^{n}(2x_i\bar{x})$$

or

$$\text{Variance} = \sum_{i=0}^{n}({x_i}^2) + \frac{(\sum x)^2}{n} - \sum_{i=0}^{n}(2x_i\bar{x})$$

We can further expand the term $\sum_{i=0}^{n}(2x_i\bar{x})$ as $2\bar{x}\sum_{i=0}^{n}x_i$ when can also be expressed as $2\bar{x}n\bar{x}$. Following our previous substitution of $\bar{x}$, as $\frac{\sum x}{n}$ the final term can be written as $-2\frac{(\sum x)^2}{n}$.

Which gives a final formula as

$$\text{Variance} = \sum_{i=0}^{n}(x_i^2) - \frac{(\sum x)^2}{n}$$

```
function calculate_loss3(I, C1, C2, C3, C4, C5):

    # Step 1: Compute squared norms and reshape tensors for broadcasting
    I_squared = square(I)
    C1_squared = square(C1)
    C2_squared = square(C2)
    C3_squared = square(C3)
    C4_squared = square(C4)
    C5_squared = square(C5)

    I_squared_reshaped = reshape(I_squared, [shape_of(I)[0], 1, 1, 1, 1, 1, -1])
    C1_squared_reshaped = reshape(C1_squared, [1, shape_of(C1)[0], 1, 1, 1, 1, -1])
    C2_squared_reshaped = reshape(C2_squared, [1, 1, shape_of(C2)[0], 1, 1, 1, -1])
    C3_squared_reshaped = reshape(C3_squared, [1, 1, 1, shape_of(C3)[0], 1, 1, -1])
    C4_squared_reshaped = reshape(C4_squared, [1, 1, 1, 1, shape_of(C4)[0], 1, -1])
    C5_squared_reshaped = reshape(C5_squared, [1, 1, 1, 1, 1, shape_of(C5)[0], -1])

    # Step 2: Combine squared norms
    combined_squared = add(I_squared_reshaped, C1_squared_reshaped)
```

```
combined_squared = add(combined_squared, C2_squared_reshaped)
combined_squared = add(combined_squared, C3_squared_reshaped)
combined_squared = add(combined_squared, C4_squared_reshaped)
combined_squared = add(combined_squared, C5_squared_reshaped)

# Step 3: Compute combined vector
combined_vector = add(I, C1)
combined_vector = add(combined_vector, C2)
combined_vector = add(combined_vector, C3)
combined_vector = add(combined_vector, C4)
combined_vector = add(combined_vector, C5)
combined_vector_squared = square(combined_vector)

# Step 4: Compute mean squared of the combined vector
mean_squared = mean(combined_vector_squared, dimension=-1)

# Step 5: Compute final result
final_result = 1 - sum(sqrt(combined_squared) - mean_squared)

# Step 6: Return the final result
return final_result
```

Whilst this rewrite appears more complicated, it is worth considering that each element of x is in fact a multi-dimensional array with size B in the i^{th} dimension feature size in the $n+1^{th}$ dimension. Thus,$Variance$ is an array of size B^n and our training loss is calculated with cross entropy on the diagonal samples n dimensional B. This is a significant step, not least for the n-dimensional loss, but because in this rewrite the whole array can be calculated without instantiating an array of size $nF(B^n)$ in memory, thus speeding up calculations. To explore why this is impractical, using the batch size of $B = 16$ or 2^4 and $n = 4$ or 2^2 and a feature space of $F = 256$ or 2^8, in this example which is modest by most production methods, the array size is 2^26 or 67 million floats being allocated just to a single term in a calculation, prohibitively large and that is with very modest (and easy to calculate numbers). Whereas, with the rewritten formula, the largest arrays present are a single B^n array, which means that for the same numbers, only a quarter of the memory is used as a single operand. With many ML library optimisations and runtime compilations to preserve a backward pass, this calculation can happen in under 100ms when optimised with in-place operations.

5.5.3 Geometric Methods for Cosine Similarity

Cosine similarity, the function being approximated with more terms, can be considered the dot product of individual elements between 2 normalized arrays. In this section, the 2 arrays to be compared will be denoted as $jandk$. When the arrays are normalized, the sum total should be 1 if they are identical. This is to say, if each component of the array were plotted on perpendicular axes, such that each point had (x,y) coordinates (j_i, k_i), if cosine similarity were 1, all points would lie on the line defined by $x = y$.

Cosine similarity can therefore be considered a measure of deviance from this line. In some respects, the comparison between j and k can be considered as a function of how far j_i, k_i deviates from the line defined by $y = x$. If the array m is denoted as the mean of j and k such that $m = \frac{j+k}{2}$, the distance between j and k can be considered in

Number	Name	Equation $S =$
1	$S = 1 - SST$	$1 - \sum_{i=0}^{n}(x_i^2) - \frac{(\sum_{i=0}^{n} x_i)^2}{n}$
2	1-L2Norm	$1 - \sum_{i=0}^{n} \sqrt{((x_i^2) - \frac{(\sum_{i=0}^{n} x_i)^2}{n})}$
3	Cosine similarity of means	$\frac{\bar{x}^2}{\sqrt{\sum_{f=0}^{F} \bar{x}_f^2}}$
4		$1 - \frac{\sum_{f=0}^{F} \sqrt{\sum_{i=0}^{n}(x_i^2) - \frac{(\sum_{i=0}^{n} x_i)^2}{n}}}{F}$
5		$1 - \sqrt{\sum_{f=0}^{F} \sum_{i=0}^{n}(x_i^2) - \frac{(\sum_{i=0}^{n} x_i)^{2\,2}}{n}}$
6		$1 - \sum_{f=0}^{F} \sqrt{\sum_{i=0}^{n}(x_i^2)}$
7		$1 - \sum_{f=0}^{F} \sum_{i=0}^{n}(x_i^2) - \frac{\sum_{i=0}^{n} {x_i}^2}{n}$
8		$1 - \sqrt{\lvert \sum_{f=0}^{F} \sum_{i=0}^{n}(x_i^2) - \frac{\sum_{i=0}^{n} {x_i}^2}{n} \rvert}$
9		$1 - \sum_{f=0}^{F} \lvert \sum_{i=0}^{n}(x_i^2) - \frac{\sum_{i=0}^{n} {x_i}^2}{n} \rvert$
10		$1 - \sum_{i=0}^{n}(x_i^2) - \frac{\sum_{i=0}^{n} {x_i}^2}{n}$
11		$\sum_{f=0}^{F} \frac{\frac{\sum_{i=0}^{n} {x_i}^2}{n^2}}{norm(\sum_{i=0}^{n} x_i)}$
12		$\sum_{f=0}^{F} \frac{\sum_{i=0}^{n} {x_i}^2}{\sqrt{\sum_{f=0}^{F} (\sum_{i=0}^{n} x_i)^2)}}$
13		$\sqrt{\sum_{f=0}^{F} (\sum_{i=0}^{n} \frac{x_i}{n})^2}$
14		$\sum_{i=0}^{n} \sum_{f=0}^{F} x_i \times \frac{\frac{\sum_{j=0}^{n} x_j}{n}}{\sqrt{\sum \frac{\sum_{j=0}^{n} {x_j}^2}{n^2}}}$
15		$\sum_{i=0}^{n} \sum_{f=0}^{F} x_i \times \frac{\frac{\sum_{j=0}^{n} x_j}{n} - x_i}{\frac{\sum_{j=0}^{n} x_j}{n} - \sqrt{\sum_{f=0}^{f=F} x_{if}^2}}$
16		$\sum_{f=0}^{F} \frac{\sum_{i=0}^{n} x_i}{n} - \lvert \sqrt{n \times \bar{x}^2} \rvert - \sqrt{\sum_{j=0}^{n} x_j^2}$
17		$\sum_{f=0}^{F} \sum_{i=0}^{n} (\frac{\lvert x_i \rvert}{n}) + \sqrt{n \times \sum_{i=0}^{n} (\frac{\lvert x_i \rvert}{n})^2} - \sqrt{\sum_{i=0}^{n}(x_i^2)}$
18	Einsum Approximation	

Figure 5.9: A Table showing the different similarity measures explored in this work

terms of m as $Distance = \sum |m_i - j_i| + \sum |m_i - k_i|$. This has been written verbosely to emphasize how this may scale in $n = 3+$ dimensions. In this sense, as n increases, the formula can be rewritten as $Distance = \sum m_i - \text{Variance}_i$

Geometrically, the difference can also be expressed as other functions such as: the imbalance of the (j_i, k_i) coordinate as expressed as area relative to the mean squared; the change in hypotenuse of j_i, k_i or the deviation from the mean. The comparative gradient of this quadratic in $n = 2$ dimensions, but scales better to more terms than cosine similarity. How the $\text{Volume}_i : \text{Variance}_i$ ratio changes with n is easier to scale than cosine similarity.

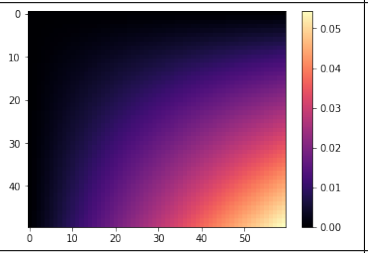
Geometrically, this can be considered as the mathematical difference between 2 squares, comparing the hypothenuse of the $mean^2$ to the hypothenuse of the rectangle defined by (x, y), or as $(m + c)(m - c)$. The equation itself appears to scale to n dimensions, reliant on the mean and the norm to compute the difference in hypothenuse of the cuboid of shape (x, y, z) and the area of the cube with an equal sum of side lengths. In the above equation, this has been rearranged to be calculated by $\text{mean} - (\text{norm(sides)}/2 - \text{mean(sides)}^2)$ which should always be positive.

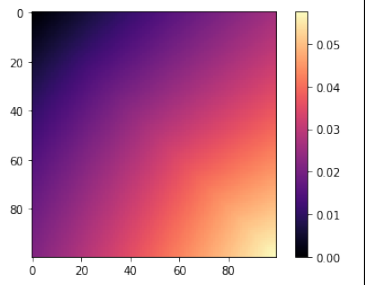
A closer approximation to the cosine similarity values is the following code that compares the hypotenuse of a cube to that of the cuboid as a function of mean values.

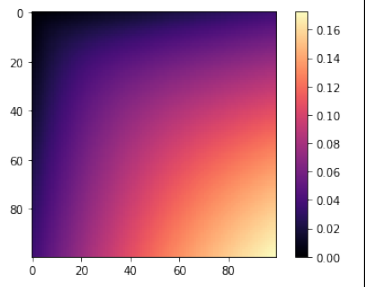
```
function compute_z(x, y):

    # Step 1: Initialize an empty matrix z with the shape (x.shape[0], y.shape[0])
    z = empty_matrix(x.shape[0], y.shape[0])

    # Step 2: Loop through x and y
    for i from 0 to number_of_rows(x):
        for j from 0 to number_of_rows(y):

            # Step 3: Calculate the mean of x[i] and y[j]
            mean = (x[i] + y[j]) / 2

            # Step 4: Calculate the norm (Euclidean distance) of the vector [x[i], y[j]]
            vector_norm = norm(vector([x[i], y[j]]))

            # Step 5: Calculate the square root of 2 times the square of the mean
            mean_squared_term = sqrt(2 * (mean^2))

            # Step 6: Compute deltaH
            deltaH = vector_norm - mean_squared_term

            # Step 7: Compute the value for z[i, j]
            z[i, j] = deltaH - mean

    # Step 8: Return the matrix z
    return z
```

By not relying on the expansion of the difference of 2 squares, a phenomenon only found in binomials, this implementation should scale better. A natural question from this formula is why dH and mean are subtracted and not as a fraction. Firstly, the mean is limited to less than one, so any division or multiplication may cause problems with values approaching 0. Secondly, the upper bound of the mean is the case where the deviation from the mean is equal to the mean, the longest hypothenuse achievable is therefore twice the mean, an upper bound of which is 1. This means z has the range $0 < z < 1$.

In additional dimensions, the formula can be modified to work for $n = 3+$. This

Image	Source
	A figure showing the plot of cosine similarity between x and y

	An alternate method generated with
	`function compute_z(x, y):`
	`  # Step 1: Initialize empty matrix z`
	`  z = empty_matrix(x.shape[0], y.shape[0])`
	`  # Step 2: Loop through x and y`
	`  for i from 0 to number_of_rows(x):`
	`  for j from 0 to number_of_rows(y):`
	`  # Step 3: Calculate the mean`
	`  mean = (x[i] + y[j]) / 2`
	`  difference = abs(( (x[i]^2 + y[j]^2) / 2 ) - (mean^2))`
	`  # Step 5: Compute z[i, j]`
	`  z[i, j] = mean - difference`
	`  return z`

	A similar plot using the difference between arithmetic and geometric mean generated by
	`function compute_z(x, y):`
	`# Step 1: Initialize z with shape (x.shape[0], y.shape[0])`
	`z = empty_matrix(x.shape[0], y.shape[0])`
	`# Step 2: Loop through x and y`
	`for i from 0 to number_of_rows(x):`
	`  for j from 0 to number_of_rows(y):`
	`  # Step 3: Calculate the mean`
	`  mean = (x[i] + y[j]) / 2`
	`  # Step 4: Calculate the norm [[x[i], y[j]]`
	`  vector_norm = norm(vector([x[i], y[j]]))`
	`  # Step 5: Calculate the square root of 2 times the ↩`
	`  square of the mean`
	`  mean_squared_term = sqrt(2 * (mean^2))`
	`  # Step 6: Compute deltaH`
	`  deltaH = vector_norm - mean_squared_term`
	`  # Step 7: Compute z[i, j]`
	`  z[i, j] = deltaH - mean`
	`return z`

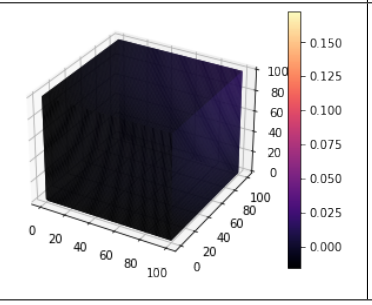
	This plot shows how cosine similarity looks in $n = 3$ dimensions on the same colour scale, thus showing the need for an approximation that scales with the same gradient throughout as $n = 2$

Table 5.2: A Table showing how cosine similarity is plotted in higher dimensions, alongside other approximations

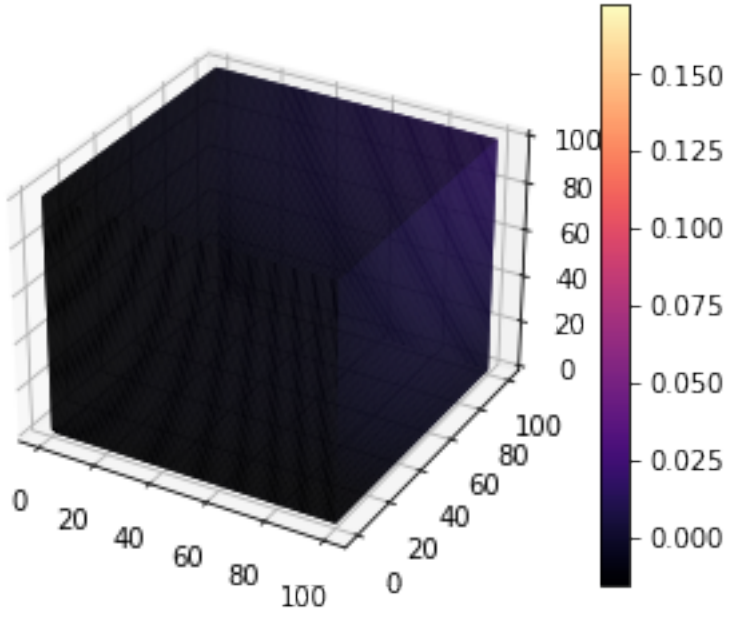


Figure 5.10: Adding an extra dimension to the cosine approx plot

modification is conducted with pythagorean formula for finding the hypothenuse in cartesian space.

5.6 Testing Method Scaling

5.6.1 Hypothesis

Thusfar, the move to n-dimensional approaches has been treated as having a linear effect on scaling. In this section, the hypothesis that this scaling is linearly related to n is tested.

5.6.2 Proof of Concept Method

Following discussion in Chapter 2, a simple proof of the efficacy of contrastive training is to consider the abilities of linear probes on simple embedding spaces. The method to test the Similarity metrics is to use these evaluation metrics as a simple model comprised of an embedding space and 2 linear layers, with noise used in between. By picking the embedding from a batch and adding noise per dimension, the test will replicated correlated inputs across modalities. During validation, the model classifies

an embedding plus noise. This approach will test the above methods, including a scaled stock method.

When interpreting this experiment, it is worth understanding that the first methods are variations on stock contrastive training that simply stack all tensor types into a 2D loss. This is very powerful because PyTorch supports highly efficient processing of this in such small scales - they were able to use a batch size of the entire sample space, so their optimum performance is little surprise.

This lightweight model will demonstrate that $n = 3+$ is worth further trials with larger models. Figure 5.11 plots the method against e^{score} to help accentuate differences.

5.6.3 Proof of Concept Results

Figure 5.11 clearly shows that different methods that calculate the variance between items n, some clearly favour certain values of n over other methods. $(n = 4)$ is the best case for Methods 10 and 16. Although 13 seems equally strong for everything above $n = 2$ where it seems to be consistent with other methods, with method 7 only working for $n = 3$ or $n = 4$. The main conclusion of this graph is that the higher values of n seem to have a minimal impact and tend to each other, which may demonstrate the limitations of this environment.

This interpretation must be balanced with the caveat that, over an infinite number of dimensions, this task becomes trivial because the noise averages 0. However, the same can be said for cross-modal contrastive training that over enough modalities the sentiment becomes clear.

5.7 Exploring Labels and Loss in $n > 2$

Loss and how this is calculated on a six-dimensional cube of logits are crucial to adding additional dimensions to the calculations.

CLIP uses cross-entropy loss, allowing each slice of the logits to have a value to maximize while minimizing the row. with labels generated as

```
labels=torch.arange(len(batch))
```

The formula for loss is given as

$$l(x, y) = L = l_1, l_2 ... {l_n}^T$$

where

$$l_n = -\sum_{C=1}^{C} w_c log \frac{exp(x_{n,c})}{\sum_{i=0}^{C} exp(x_{n,i})} y_n, c$$

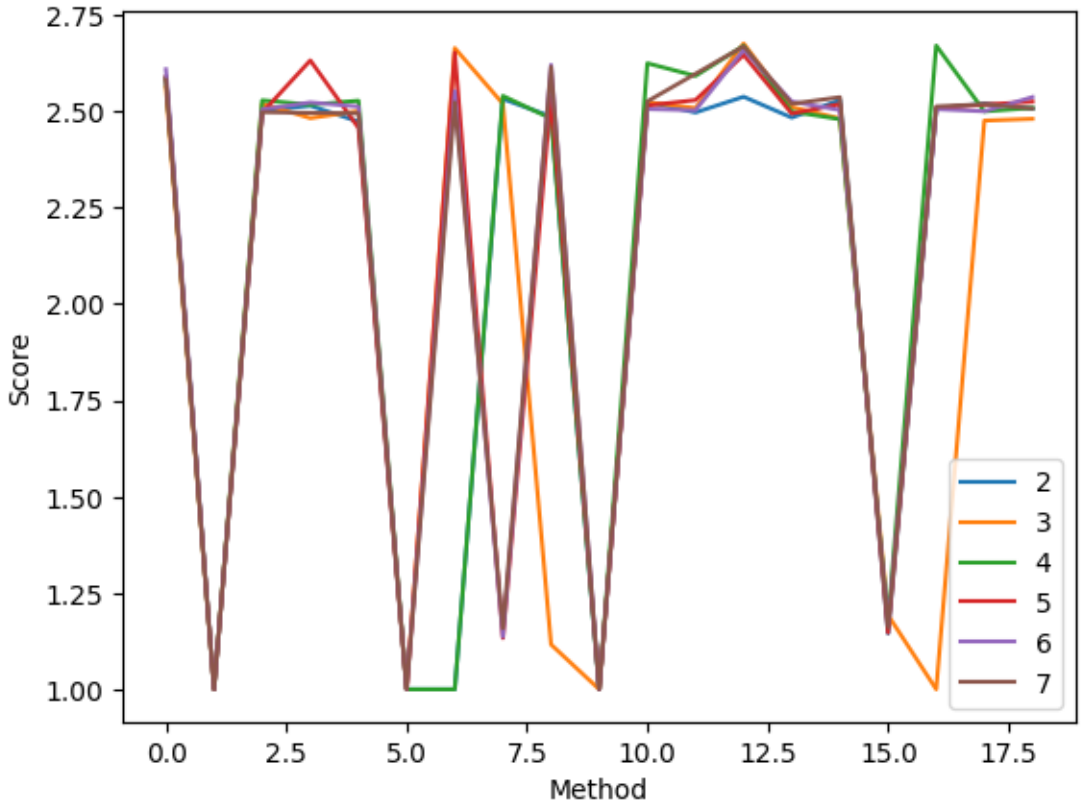


Figure 5.11: Method performance in each value of n, scored by the exponent of the ratio between positive and negative logits. Scores near 1, represent models unable to converge, with higher scores representing sufficient gradients able to converge. Notably, some methods (6-9) viability differs by number of dimensions.

This formula means that for a given value of x and target y, the loss is the logarithmic expression of the exponent of the correct class, over the sum of exponents of all other classes, which is then balanced according to any class or output biases and summed over the batch. However, the implementation for 2 dimensions receives a target of indexes of type 'torch.long'. However, in the additional dimensions, there exist values that are not in the same row, which must still be considered for loss. Recalling the initial figure, the front face must be considered when maximising the value at $(0, 0, 0)$. Thus, for n dimensions, using a class list in $n - 1$ dimensions is insufficient: there are many locations where a class to maximise is misleading, and we would only consider a single dimension. Instead, a facsimile of the initial figure is created by diagonally embedding the ones into a cube of the correct dimensions. This offers a more effective loss metric.

Contrastive training in 2 dimensions is capable of treating a square of logits as a stack of classifications such that l_n represents the loss for a given row where x is the row and y the IDX of the row, and thus the desired item to calculate the loss of. using a transpose for the calculation of P(Image|Caption) and subsequently, PCaption|Image. However, in 3+ dimensions, the cube has additional properties. It is no longer correct to say that for a given logit, it should either be Positive or Negative as may be the case with Cosine Similarity.

In higher dimensions, we rotate through the dimensions. This has 2 advantages: It does not require recalculation and every dimension occupies every position, eliminating any error introduced by the randomness of exact labels as seen in Table 5.3.

In each loss metric, consider all encoder outputs as X where X_i is the output of the encoder i. The plane corresponding to the batch output X_i is a Tensor of similarities that each element of X_i matches every other encoder output. In a six-dimensional matrices of dimensions B^n, the item at index (i, j, k, l, m, n) can be considered probability $P(X_{0i}|X_{1j} + X_{2k} + X_{3l} + X_{4m} + X_{5n})$, bases on the original assertion, that each view of the loss matrix, is the probability of a particular encoding, given all others in that slice of an n-dimensional matrix. This approach represents an upscaling of the original CLIP paper proposing each logit represents the probability of an image, given a caption, and vise versa in the transpose.

This contrast to sigmoid loss is used for the majority of experiments due to the fact that sigLIP was predated by some years with this implementation. However, there is functional equivalence between the element-wise CELoss and sigmoid approach.

5.7.1 Changes to Implementation

Original code from the initial CLIP release pointed to the use of cross-entropy loss as taking labelled entries. In additional dimensions, this compares the matrix at

that label with all others, essentially projecting a slice of a $(n-2)$-dimensional space against other slices.

This is equivalent to projecting the diagonal into each dimension in the cube, resulting in a paradigm that seeks to maximise and minimise at different stages of the loss gradient. The intuitive answer is to argue that if only a single transformers gradient is considered, then it would mathematically simplify to correcting the vectors and moving incorrect ones further away. However, it is unclear whether this is happening in such a case; this assumes that all encoders are resolved independently, which would require the pairwise approach that has already been discussed as being impractical due to the adverse scaling of the combinatorics.

This becomes very pertinent as the dimensions and limits of this approach are explored. The combinatorics in high dimensions are perplexing, but drastically increase the chance of a given logit being maximised.

The chance of any logit being maximised by a loss function is the likelihood in a given size B in n dimensions, that the position is the same in any dimesion.

$$P_{maximized} = 1 - \frac{\prod_{a=0}^{n} max((B-a), 0)}{B^n}$$

The logic underlying this is the reason the implementation takes a set of labels of identical shape to the input, as a way of having increased control over the logits' training.

5.7.2 Limits of the Function

There are multiple factors to consider when reproducing this: First, in 2 dimensions, the limit to this function is in the range -1, 1. This can be shown geometrically and is the purpose of the dividing by the norm. In other words, the magnitude of each vector is divided by its hypotenuse, which means that each input vector is of size 1.

An initial suggestion would be to mean the component vectors. Taking the maximum and minimal values results in the correct range; however, there is the possibility that the mean is 0 in magnitude. In this case, dividing by the magnitude of the mean to get a hypothenuse of 1 is not possible.

In all this, it must also be considered that a distinct gradient descent must be possible - There can be no false minima to the resultant function. The goal of this work is not just about finding a function that meets the criteria of the perfect case but also allows a sloping gradient to the minima.

Against this, we must also consider the gradient of the mathematical function; even if we converge and have a sloping parameter gradient in a high-dimensional space, we need to ensure that the whole space is encompassed in this slope. Several experiments are used to isolate regions with a mask to show the effect of loss. It is easy to see

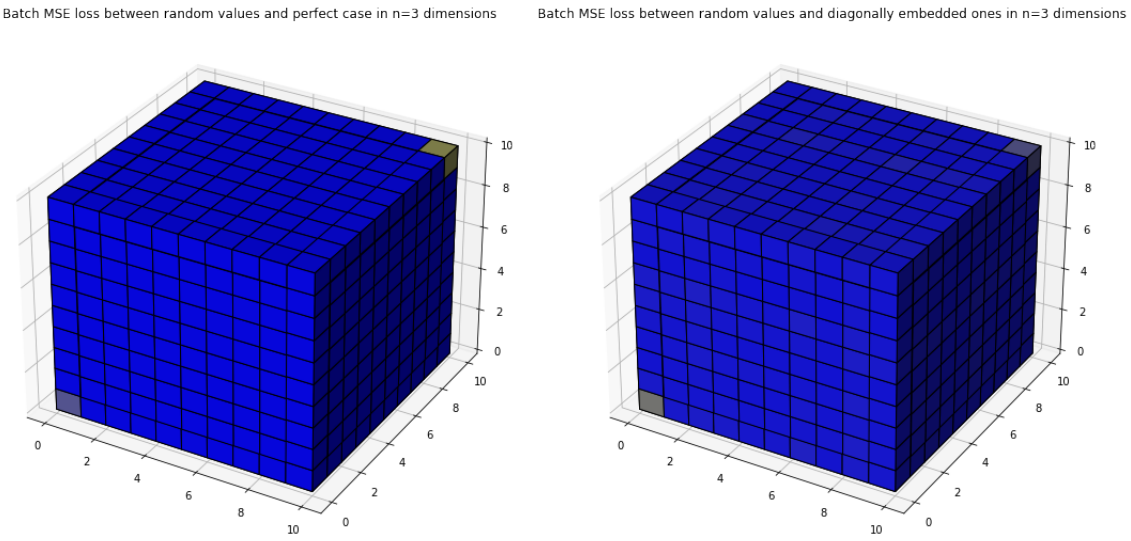


Figure 5.12: Difference between using example Loss and generated labels

how these regions vary from the corresponding labels with MSELoss, but with cross-entropy that scales better to magnitudinal variation this requires a different approach. Curiously, there appears a significant difference between the following graphs: Figure 5.12 shows how these perfect labels are generated and understanding the merits of this system. In a perfect case, the logits in the regions where $m < n^2$ or the case where not all coordinates are the same, the similarity should be minimal. In the training case where the logits are essentially random within the initial steps, this similarity should also be minimal, so the training for these initial steps is negatively impacted by the loss being averaged across these regions.

In spite of predicting huge performance boost from additional data, in initial tests, the result was that with $n = 2$, the linear probes converged to around 55-60% accuracy within 3000 steps. With the same parameters, $n = 6$ required 12k steps. Understanding the impact of subregions on training yields an answer to this behaviour.

Now that each region is isolated, 3 approaches exist: using a mask, using exact weightings, or a learnt parameter to switch between the training paradigms as presented in a study on robust representations where a scaling factor is used between pre-trained and novel methods [122].

The merits of using weightings can be shown in graphs like the following. These show the ratio of MSE Loss that each region accounts for. There are several distinct regions in each graph, and notably that 2 graphs are inverted, corresponding to the most varied regions, indicates two distinct behaviours, especially as these 2 regions are the regions expected to be near-perfect at the commencement of training.

It is reasonable, therefore, to consider that the 3 stages of training are as follows:

- Maximizing matching items
- Minimizing non-matches
- Maintaining equilibrium

5.7.3 Label Smoothing

An approach found to improve teacher-student supervision-based approaches has been the use of label smoothing on the teacher. This modifies the typical CE loss approach to take a parameter T such that the loss is calculated on the CE loss(logits/T). As T tends to be infinite, the loss becomes smoother, resulting in labels that are equally nonzero for nondesired classes. In our use case, this would look like the modification of one-hot ranges of labels over our n-dimensional cube to a similar distribution but a near-one-hot with 0.99 and the remainder of the norm split between non-optimum classes. This approach should be considered to have the same effect as is found in semi-supervised approaches of altering the loss from non-desired classes to be truly minimized rather than ultimately ignored in many cases.

The expectation of this approach to label modification is a continuation on the previous discussions on differing loss metrics, which is to further force reductive behaviours for non-maximal values.

A drawback of using very precise "exact" labels generated with a perfectly identical case is the highly improbable case where in spite of input variability, a very particular vector is learnt. Instead of just avoiding this exploration, other cases can be explored, such as removing any intermediate labels between extremes, which carry a large number of values. It is worth considering the contribution to loss of these regions with partial matches, and in many cases, they are not the main contributors. They have the largest quantity of values and some of the least input per value and have a high amount of instability.

There are 4 approaches trialed, which appear as "exactlabels", "maskLosses", "normlogits" and "Loss" respectively in later demonstrations of hyperparameter sweeps:

- Cross Entropy Loss, using the as written implementation for $n = 2$ that only maximises the diagonal
- Partially masked CE Loss, removing problematic regions, emphasising a 2-region approach
- Scaling loss by region distance

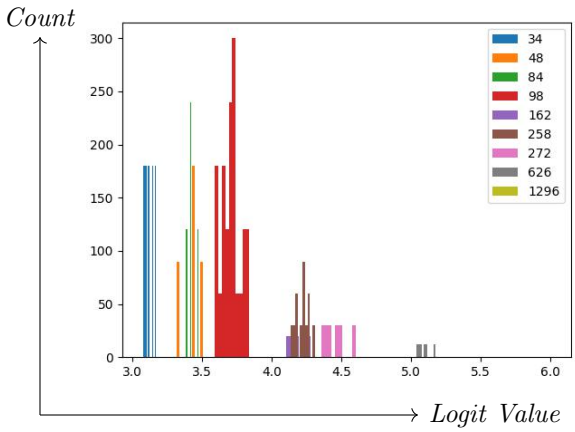


Figure 5.13: This plot shows the values (x-axis) and the frequency (y-axis) of different regions where $B = 4$ and $n = 6$. This shows that although each region is distinct, the values each should hold have high variance

- Sigmoid Loss, scaling all labels and values to the [-1,1] range

These approaches are explored in subsequent experiments. The impact these each have on the way logits perform is demonstrated by the distributions taken from when exact labels are used for method 8 with $n = 6$. Shown in Figure 5.13, the logits collected have a distinct distribution for each parameter set.

The region corresponding to 98 is the largest to maximally enforce loss, but if this label is reachable with a random vector in many cases, there is a low probability, and this region will greatly contribute to loss, especially with such high variance in labels.

If it were to be masked, there are still significantly more logits available to training that this work would still present an improvement on scaling than traditional methods. Supposing that this could be done faster than currently implemented, a large amount of the computational graph could be saved.

The final approach, however, is to use the density of this region to influence the gradient step: any significant deviation from this label is arguably far more impactful to training than either of the other label distributions, therefore weighting loss by the mask density is a strong way to stabilise training.

When replicating the advances of sigmoid loss in SigLIP in high dimensions, there is a comparable performance improvement.

Cross-Entropy (CE) loss factors in the exponent of every item, not just the corresponding probability.

$$l(x, y) = l_1, l_2 ... {l_B}^T$$

where

$$l_b = -\sum_{c=1}^{C} w_c \log \frac{\exp x_{b,c}}{\sum_{i=0}^{C} \exp x_{b,i}} y_{b,c}$$

In the original training, CE loss is taken in both directions, maximizing the positive class against others, giving each encoder a loss function. These are then averaged to represent the overall loss. On the grid of logits, this compares each item on the diagonal to its corresponding row and column.

However, in understanding that there were only two regions in that training, scaling that to additional dimensions presents a conundrum of calculating the loss for each dimension.

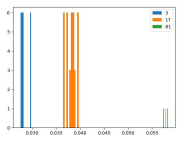
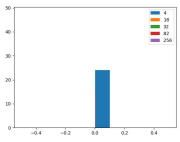
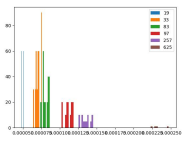
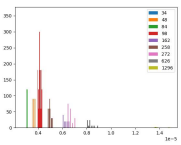
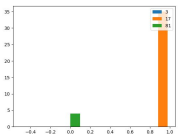
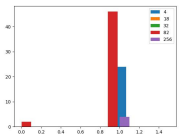
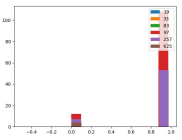
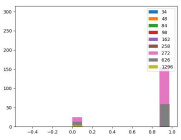
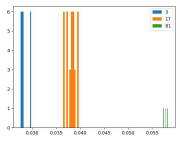
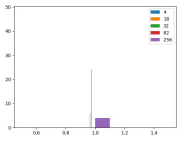
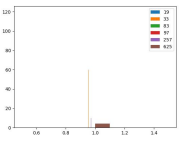
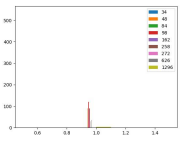
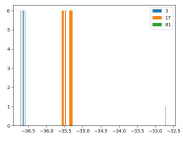
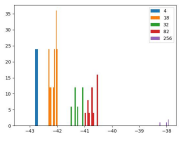
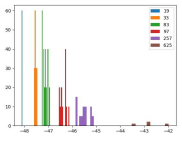
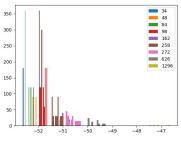
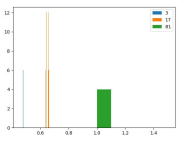
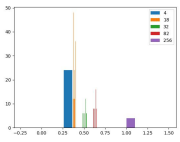
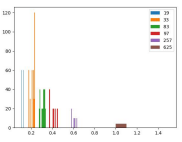
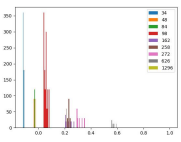
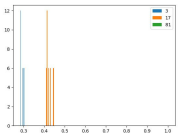
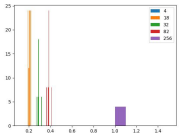
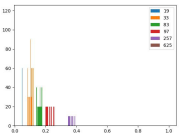
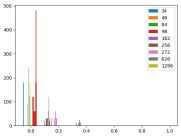
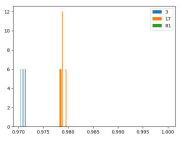
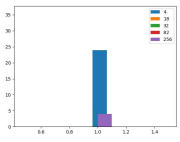
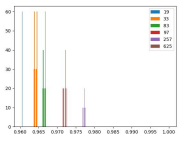
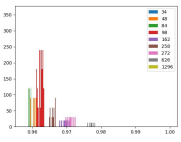
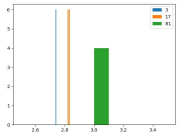
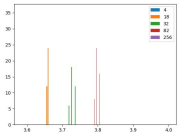
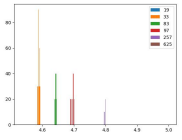
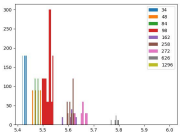
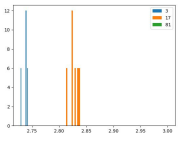
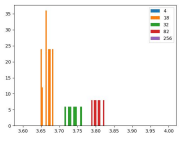
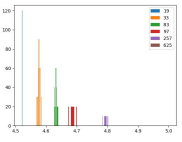
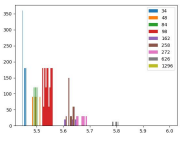
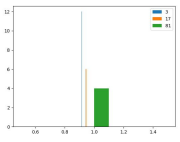
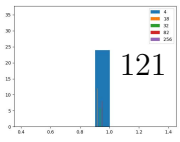
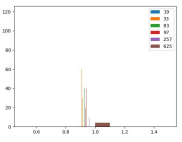
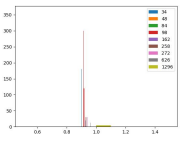
5.7.4 Labels Exploration

Many of the above methods have varying behaviours, values and tend to different shapes during training. In 512-space, the overall topology is what is relevant, rather than the absolute values, as is true for Loss calculation: the length of step towards an optimal parameter set has no bearing on the location of representations. For many experiments, using exact labels has been exhaustively trialled: working out the correct labels for areas of logits based on a single test case. Rather than contorting to the arbritrary labels of a 2-region system like CLIP, we use this to demonstrate why scaling into additional dimensions comes with overhead and difficulty.

Table 5.3 is a key tool in identifying poor algorithms that produce labels with differing results or muddled orders. Cross-referencing with Figure 5.11 helps identify the primary methods by their loss characteristics, and the trends as n increases become apparent. In particular, this assumes the unique case in which all encoders output, F identical tensors that occupy a uniform distribution across the range $0 < \mathrm{F}_i < 1$. This is not a given, and a Gaussian distribution would be a better predictor of labels - this was initially discounted on the assumption that the imbalance in masked regions might stop the model converging properly.

Instability is also caused where the regions have a relatively high variance, as this can cause converging issues. The best explanation for this is that the labels are affected by input value: The test to generate these labels uses random, but identical, tensors. Thus for there to be variance in the labels reflects that the individual values affect the output rather than the similarity to adjacent points.

Compared to SigLIP as previously discussed, many of the similarity comparisons are not bounded between -1 and 1. For a fair comparison of this method, when sigmoid loss is tested, the logits are scaled down to the correct range prior to calculation. Sigmoid loss should perform well in the cases of both exact and theoretical label values. The significant advantage of using sigmoid loss is that the values are individualy not constrained by the logits adjacent. To preserve the effectiveness of CE training, the

Method	3	4	5	6
0				
1				
2				
3				
4				
5				
6				
7				
8				
9		 121		

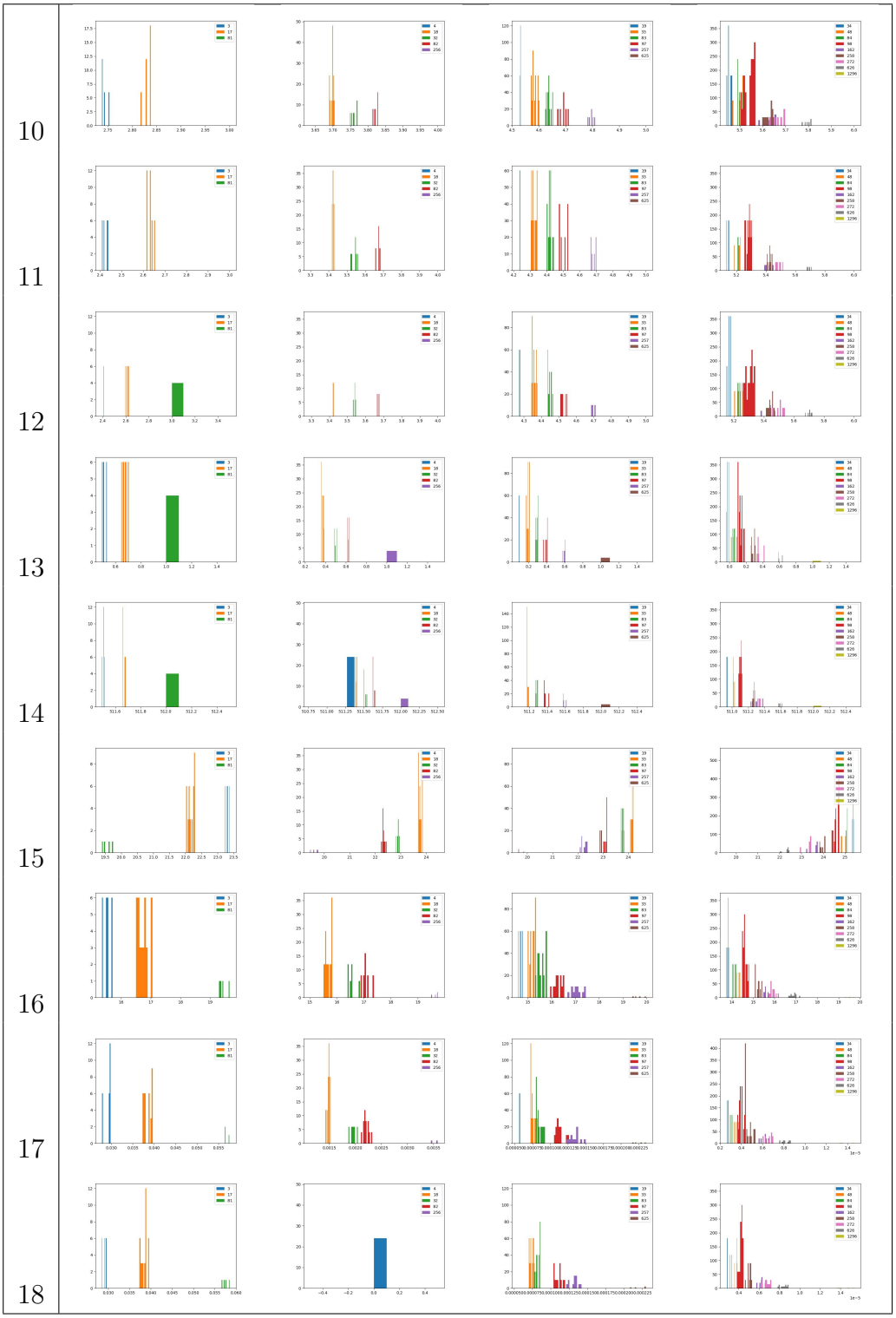


Table 5.3: Distribution of labels by region when using exact labels

logit cube is permuted so that logits are compared against each dimension. This permutation is left for the purpose of fairness in training and not modifying the wider framework. Comparison in this way allows the sigmoid approach to be trialled against a variety of label and similarity approaches to show whether this method also carries the same scaling advantages.

5.7.5 Prototyping Modalities with Projections

In this chapter, the discussion is around scaling numbers of encoders, with the emphasis on being able to support multi-modal inputs, like many Vision-Language models; the goal is to bridge multiple modalities. When the fundamentals of grounding vision and textual domains are considered, it is naive to believe that both naturally share the same semantic space given the lack of 1st-order grounding in this context.

CLIP leans on an intermediate embedding for cross-modal comparison. This is essential as there are some sentiments in a visual domain that are difficult to linguistically describe. Similarly, there are myriad textual expressions that have no visual equivalent. This is the conversion between different modalities whilst preserving the backward gradient, crucial to the ability of embedding spaces to be adapted towards carrying information.

At a low level, this means that the function for comparing text to image, defined as f such that $f(T, I) = P(T|I)$ where $T = \text{TextFeatures} \cdot \text{Projection}$. (Using $\cdot$ to represent matrices multiplication). As the projection has the shape of that latent space of text features, and image features, the final shape of T will be the batch size by the image latent space, creating a unified embedding space for cross-modal comparison.

However, since this occurs in higher dimensions, where the abstract function $f(\cdot)$ is now defined as $f(T_1, T_2, T_3, T_4, T_5, I) = P(T_1|T_2, T_3, T_4, T_5, I)$, T repeatedly including the textual projections may be problematic. As discussed above, conflicts between polarities may occur. This would be exacerbated by a $n - 1$ terms including the projection in the calculation. In an ideal case where the n operands are identical, the output of $f(\cdot)$ will be created by the product of features that will take one positive or negative $(+ve| - ve)$. Given feature 'x', projection for the feature (p) will also take a Boolean positive or negative. This means that the polarity of the overall feature is effectively $f(xp, xp, xp, xp, xp, x)$. If we assume that $f(\cdot)$ can be abstracted to a multiplication of features, on which many subsequent methods depend, the following truth table occurs:

p / x	+ve	-ve
+ve	+ve	-ve
-ve	-ve	-ve

This truth table shows that negative values in either a single feature, or in the text embedding can drastically impact the output space. We test this in subsequent experiments by naively totaling the features across training. This indicates a rough tally of how many are negative and the general trend. The question becomes whether any such system can learn meaningful embeddings.

Alternatively, an alternative learned array $'P$ that means that in 2 dimensions $f(t \cdot P, I) = f(t, I \cdot' P)$. Which then evaluates the function $f(\cdot)$ for:

$$I \cdot (t \cdot P)^T = (I \cdot' P) \cdot t^T$$

However, as this is normalised before calculation, it means that $'P$ must be derived from the following :

$$\frac{i}{\sqrt{\sum i^2}} \frac{tP}{\sqrt{\sum tP^2}}^T = \frac{iQ}{\sqrt{\sum (iQ^2)}} \frac{t}{\sqrt{\sum (t^2)}}^T$$

Derivation A: Starting with

$$\frac{i}{\sqrt{\sum i^2}} \frac{tP}{\sqrt{\sum tP^2}}^T = \frac{i'P}{\sqrt{\sum (i'P^2)}} \frac{t}{\sqrt{\sum (t^2)}}^T$$

We multiply each side by

$$(\frac{i}{\sqrt{\sum i^2}}^{-1} \frac{t}{\sqrt{\sum (t^2)}}^{-1})$$

Which gives

$$\frac{i}{\sqrt{\sum i^2}}^{-1} \frac{i}{\sqrt{\sum i^2}} \frac{tP}{\sqrt{\sum tP^2}}^T \frac{t}{\sqrt{\sum (t^2)}}^{-1} = (\frac{i}{\sqrt{\sum i^2}}^{-1} \frac{i'P}{\sqrt{\sum (i'P^2)}} \frac{t}{\sqrt{\sum (t^2)}}^T \frac{t}{\sqrt{\sum (t^2)}}^{-1})$$

This allows for:

$$\frac{tP}{\sqrt{\sum tP^2}}^T \frac{t}{\sqrt{\sum (t^2)}}^{-1} = (\frac{i}{\sqrt{\sum i^2}}^{-1} \frac{i'P}{\sqrt{\sum (i'P^2)}}^T)$$

This cannot be readily simplified further, so in lieu of being able to express the image projection as a function of the text projection, we simply allow the projection to be a similarly learnt parameter.

An alternative approach is to consider the equivalent similarity metric in the perfect case.

$$T \cdot P = I$$

Projection Value	Definition
'none'	No projection is used for any modality
'inv'	Using the inverted transform on the repeated modality
'iinv'	Using the an inverted transform on the image encoder

Table 5.4: A table describing the projection hyperparameter

This can easily become

$$T = I \cdot' P$$

However, this implies that there is an inverse for P and that the similarity between TP and I also correlates with the similarity between T and $I'P$.

Interestingly, the standard projection is initialized with a standard deviation of $\sqrt{F}$, with the majority values around 0.01e-3. This projection to smaller scales is later corrected by multiplying the exponent of the logits, which brings the maximal value back to around 1 ahead of the final calculation. This reduction in projection scale stabilizes the projection while helping to avoid infinite and null values. This scaling also enables the exponents of each logit (a key part of the loss calculation) to stabilize to optimal values, rather than having large imbalances.

```
logit_scale = torch.nn.Parameter(torch.ones([]) * np.↩
    log(1 / 0.07))
```

In summary, as a proto-type for increasingly multimodal machine learning applications, the experiments presented cannot afford to unfairly bias a particular modality, nor be skewed by the repetition of one. Therefore, a hyperparameter is introduced to experiments that takes several values.

5.8 Evaluating Regions of n-dimension Loss

To observe how this works, the table of graphs shows three very distinct stages of training based on the regions stabilizing relative to each other.

All of these graphs were generated with loss calculated from exact labels and the rough, one-hot mask of the diagonals. However, they did not contribute to the gradient during training, and despite training converging, the logit delta between good and bad that might be expected failed to grow.

Theoretically, this calculation carries the same geometry as seen in the stock implementation of CLIP. CLIP generates one set of loss per dimension, and then to be averaged together. Literature is lacking on the combinatorics of extra dimensions. An assumption can be made that the scaling of additional dimensions is equivalent to the scaling of multiple devices as seen in LLip and SigLip. The standout assumption

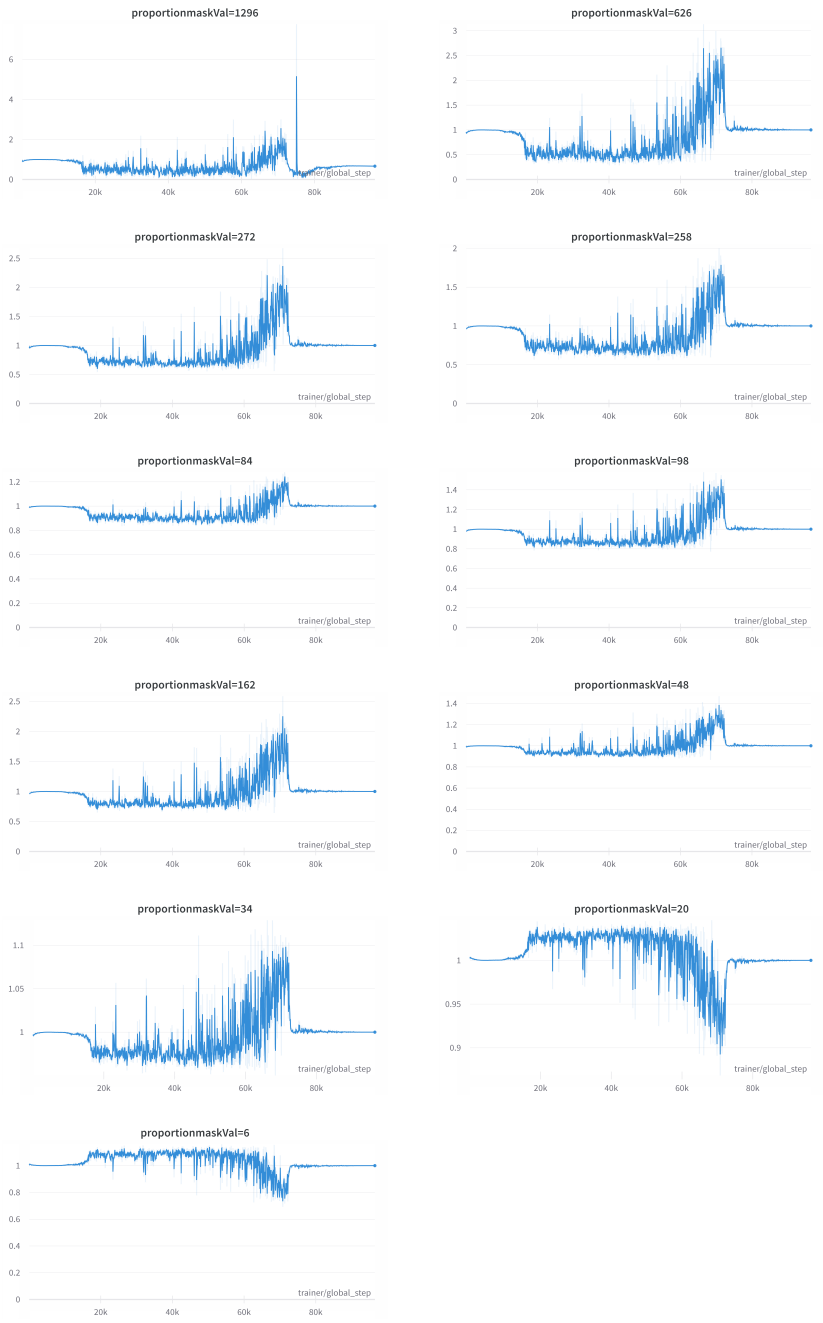


Table 5.5: A series of graphs showing that the smallest and largest regions occupy inverse areas during training and are unstable initially during training but find an equilibrium

is each work is the neccessity of every position to have been filled, and that asymetry has no lasting bias.

An immediate change to make then is to replace the transpose of arrays with a permute of all logits, rotating the cube around each face. We then substitute the probabilities to be expected at each region. When we break down the loss into regions, we do so according to the numbers of unique coordinates. That is, by the minimum expected loss per logit. This is well modelled by the concept of using exact probabilities calculated with n identical inputs and using the loss output as the ideal case.

The issue with this, when factored into calculations, is to suggest the value it tends to without necessarily solely contributing to the corresponding logits. Consider the issue with $n = 3$ dimensions. Applying the loss as seen in CLIP to each diagonal, we find points within certain regions of the cube that are never used, and this subdivides our region mask.

This subdivision suggests that some of the masks in each region do not contribute and should not be in higher dimensions. This suggests that instead of viewing $n = 2$ as the case of all or none of the coordinate matches, it is, in fact, $n-$similar or $n-1$. The $n-1$ region is the part that scales better with batch size. Indeed in higher dimensions, this continues to region obey the same scaling. The merit of adding additional regions is therefore the addition of regions that have better scaling beyond $n = 4$.

Another way of performing region masking in an attempt to model just using the speed up of regions defined as masks[-2:] is to flatten the array by selecting solely the logits defined by the masks $\text{masks}_m \forall m$. However, even when providing probabilities to cross-entropy loss, it means that the loss is additionally calculated for all other values, making each positive case more sensitive to other, uncorrelated, case outcomes.

This is why a learn mask is also tried, starting as a set of ones per mask. To stop the model minimising the regions to zero, we divide it by the norm to ensure that the magnitude of value can never approach zero, then performing a softmax operation to limit the weights to between 0, where it would be entirely removed, and 1 where entirely used.

The final approach worthy of consideration is to take the square that sits along the diagonal of the n-dimensional cube and perform loss on that. However, this would require the creation of $n-1$ losses per dimension for n dimensions. The tracking of gradients that scales with n^2 is unrealistic.

5.8.1 $n = 4$ Dimensional Comparisons

Table 5.6 shows the different methods outlined above in a range of cases, across several different methods, but mapped to the same colour scheme. Compared to the example in Table 5.2, this shows how the gradient of similarity may look when compared across B^n samples where $n = 4$. Projected into a flat grid, this demonstrates both the presence of distinct regions, that follow a repeating pattern, and the relative values each region holds in n dimensions.

To interpret the table in 5.6, consider the difference between the majority of points in the perfect case and those in the random case, they should be roughly equivalent. It is also worth noting that the input vectors have a L2 norm of 1. As explored previously, scaling into additional dimensions of loss is problematic and can introduce different behaviours. With additional dimensions, it is worth considering the extra columns this table would have. In these representations, the colour grading maxes the distinction between high values difficult, printing these graphs with normalisation graph = graph/max(graph) transform means that some methods suffer and only show a single white dot, this is not a worry considering how this manifests in use. This distance between large values helps maintain the second-order awareness of the model. That is, first items are as similar to first items as second items are to second items.

5.9 Training Results

In this section, the best performing runs will be presented, showing the successful accomplishment of **RO.2**. In this respect: much data has not been published here, but the following parts show a snapshot of the training which may go on to have potential being paired with suitable decoder training for highly effective use in low-resource or mixed-modal domains.

The following graphs should be interpreted understanding that the sets of coordinates that should match are divided such that the sets of matches s obeys the formula:

$$\sum_{i}^{i=s} S_i^6 = \text{MaskValue}$$

So for a given MaskValue, a set of numbers which sum to 6 can be worked out which when squared, then cubed, sum to the maskvalue, and represent the counts of each unique coordinate.

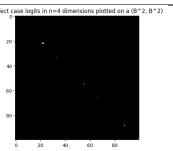
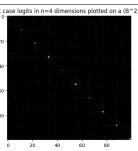
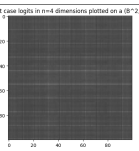
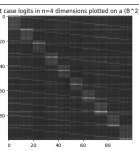
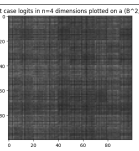
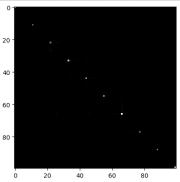
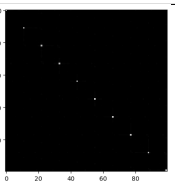
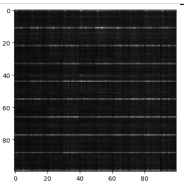
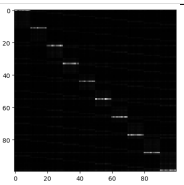
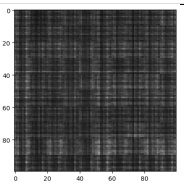
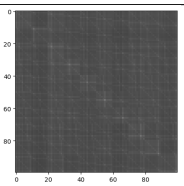
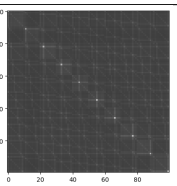
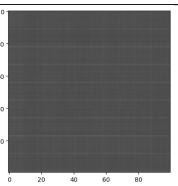
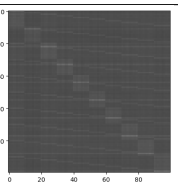
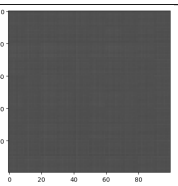
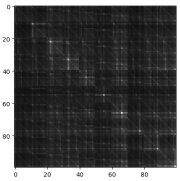
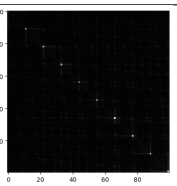
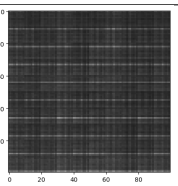
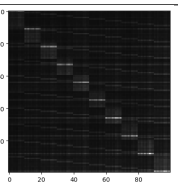
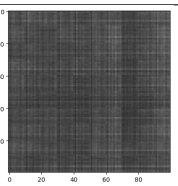
Method	Perfect Case with noise	Perfect case	2/4 perfect	3/4 perfect	Random Case
Einsum (18)					
Cartesian Distance squared Logits (2)					
Cartesian Distance (1)					
Hypothesis comparison (4)					

Table 5.6: Visual analysis of unnormalized loss functions on similar colour scales

5.9.1 Best Run (in sample 150) for 6 DIM training, 10 Epochs, MSCOCO

This run is representative of many runs, in the experiment set available publicly through W&B, using a very limited number of transformer layers, selected such that the parameters are not saturated.

n	B	Layers	Epochs	Time
6	6	3	10	15 Hrs

Table 5.7: Run Details

This particular run is highlighted, because the Linear probes, as discussed in Chapter 6 performed within 10% of next nearest competitor, but top-k=1 Imagenet score is over 30% better, suggesting a strong optimum case.

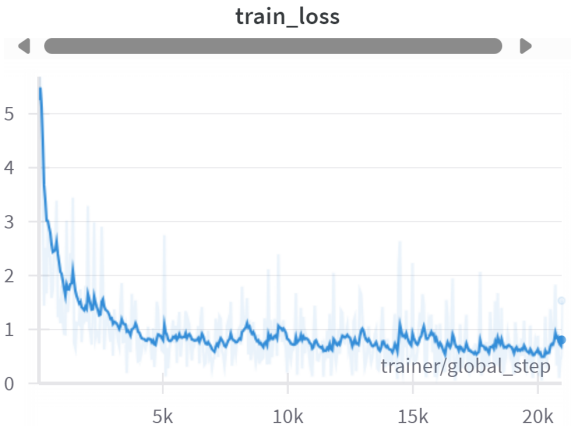


Figure 5.14: A graph showing the training loss

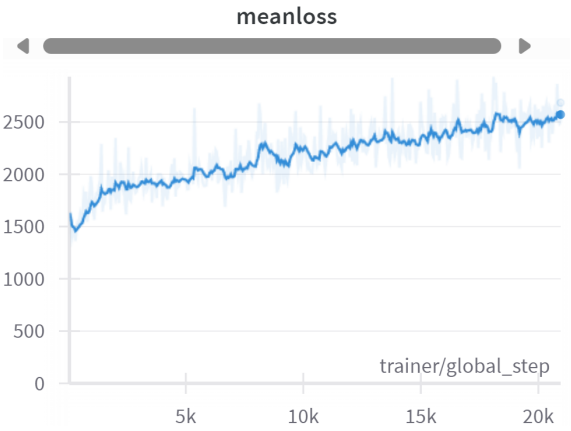


Figure 5.15: A Figure showing the mean loss metric throughout training

Figure 5.14 shows the training loss, which looks as expected, but notable is the noise, even far into the training. Further, the immediate convergence to a loss below 2.5, which contributes to an impressive performance even before the end of the first Epoch.

Figure 5.15 tracks the average value of loss in the logit comparison cube, which should be biased in favour of logits with entropic values. The growth of this value slowly, shows that the model is increasing the distance between all embeddings, which is to be expected - it would be intriguing to be able to further scale this work to see whether there are limits to this, or a point where the values exceed the limits of the system. The mean validation logits shown in Figure 5.16 is analogous to the mean

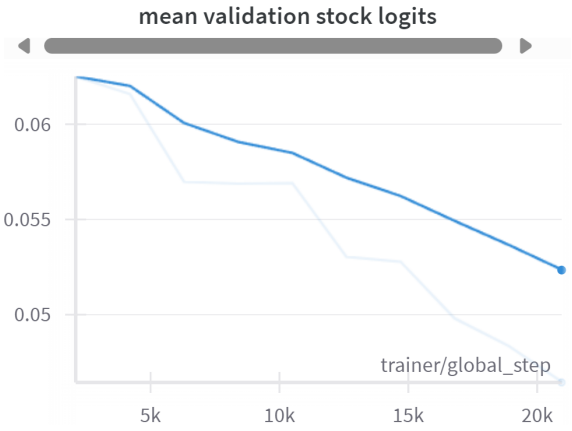


Figure 5.16: The average logits in the stock method during Validation

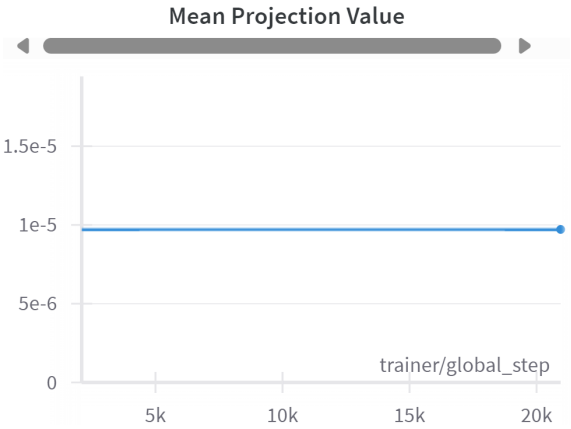


Figure 5.17: Mean projection Value

loss, only it tracks the value when performing the original matrices cosine similarity, which remains remarkably stable, showing that the poor logits are likely the driving force in the loss operation, which is a significant result.

Comparatively, Figure 5.17 shows that the projection parameter between modalities does not significantly move during training, suggesting that the model's projection between modalities is not a significant factor and does not carry a huge charge. Logit

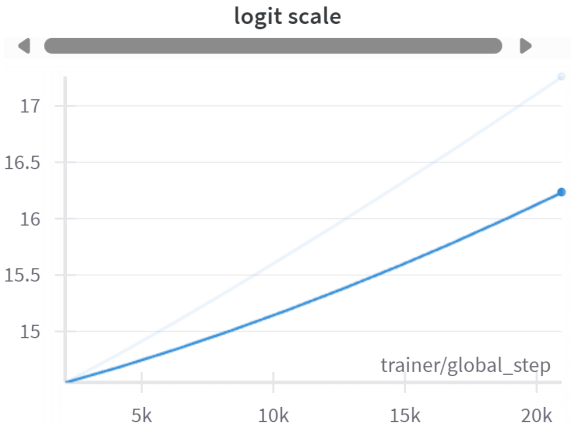


Figure 5.18: Improvement of logit scale during training

scale shown in Figure 5.18 is a measure of confidence in a model reaching a perfect case - its rate of increase correlates to model performance, and excitingly, follows the

replication of a baseline clip implementation.

The region defined as maskval = 34 is clearly a major contributor of loss, which is when there are 2 pairs of matching values and another 2 isolated ones. The convergence and relatively stable value suggest that this set of regions can never be minimized and was the inspiration for trying adaptive labeling. The region maskval = 48 contributes equally to maskval = 84. This reflects that maskval = 84 is a single set of 3, where maskval = 48 is 3 sets of 2, both challenging cases for any clustering measure. The comparative equality achieves the research objective: showing that multiple encoders can be efficiently combined for loss calculation.

Table 5.8 is plotting the proportion of loss that is accounted for by the area where all the values are unique. That this decays over training suggests that while initially these locations have a large contribution to initial embedding placement, over time, they contribute less compared to where subsets of matches occur.

5.9.1.1 Analysis

Table 5.9 shows the comparative MSE values away from labels within the logit cube produced in 6 dimensions. The interesting values to observe are the graphs maskval = 6 and maskval = 1296 which cover the cases where the index within the batch are all completely different and the case where they match, respectively. The difference in these graphs shows how in the case where they are all different. The notable features on these graphs are the comparative scales. The case where no match is present is initially true and does not get very large during training. Comparatively, the case where items are meant to match becomes very high in magnitude and starts further from the labels than maskval does. This states that the labels are far from achievable for the model to descend to, but also that this region is responsible for the largest gradient input. Further notable is that for all values greater than 20 (every case where at least something matches) there is an 'elbow' in the graph at around 2k steps. This suggests that this is the minimum number of samples to begin learning from. The observation roughly corresponds to a single epoch of training. Given the training sets are shuffled, this peak existing demonstrates the power of this method, that the model is reevaluating the prior representation compared to the new batch of data.

The plots in Table 5.8 show that at this elbow, the regions stabilize. The proportion of MSE loss by region becomes relatively fixed, with the closer to perfect cases accounting for increasingly large chunks. This perhaps is to be expected by the imbalance in size of each region within an n-dimensional cube. This imbalance being the potential cause means that there is a subsequent hypothesis that this training method may have an upper bound of batch size, where the scaling laws that this work is testing may no longer apply. Unfortunately, this is beyond the bounds of what is testable at this scale.

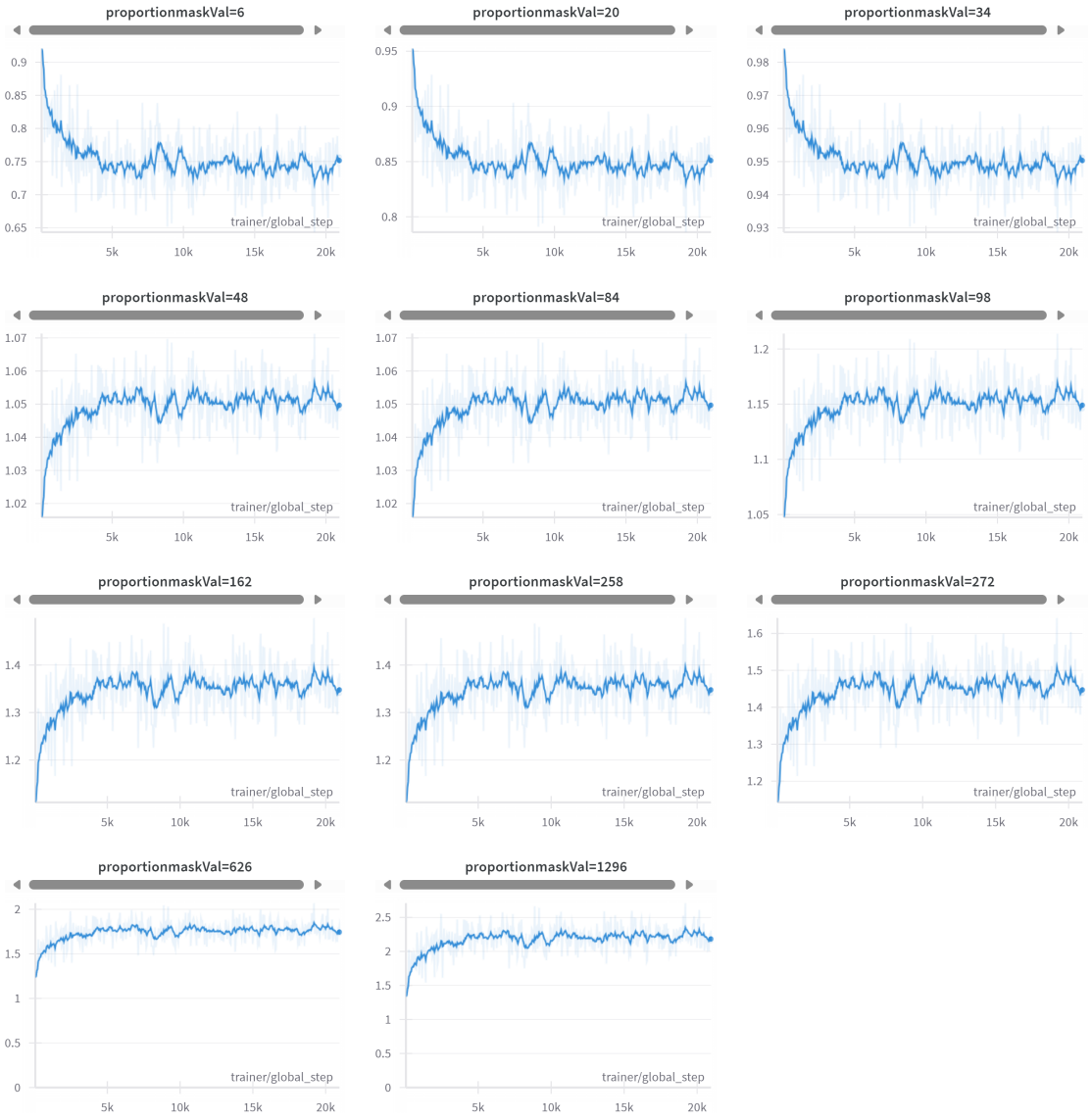


Table 5.8: A table of figures showing the proportion of loss that each region accounts for. Notably, the graphs appear to flip around maskval = 48, which represents 3 pairs of matching coordinates. From, which point, the regions where points cluster together, account for the largest proportion of the gradient.

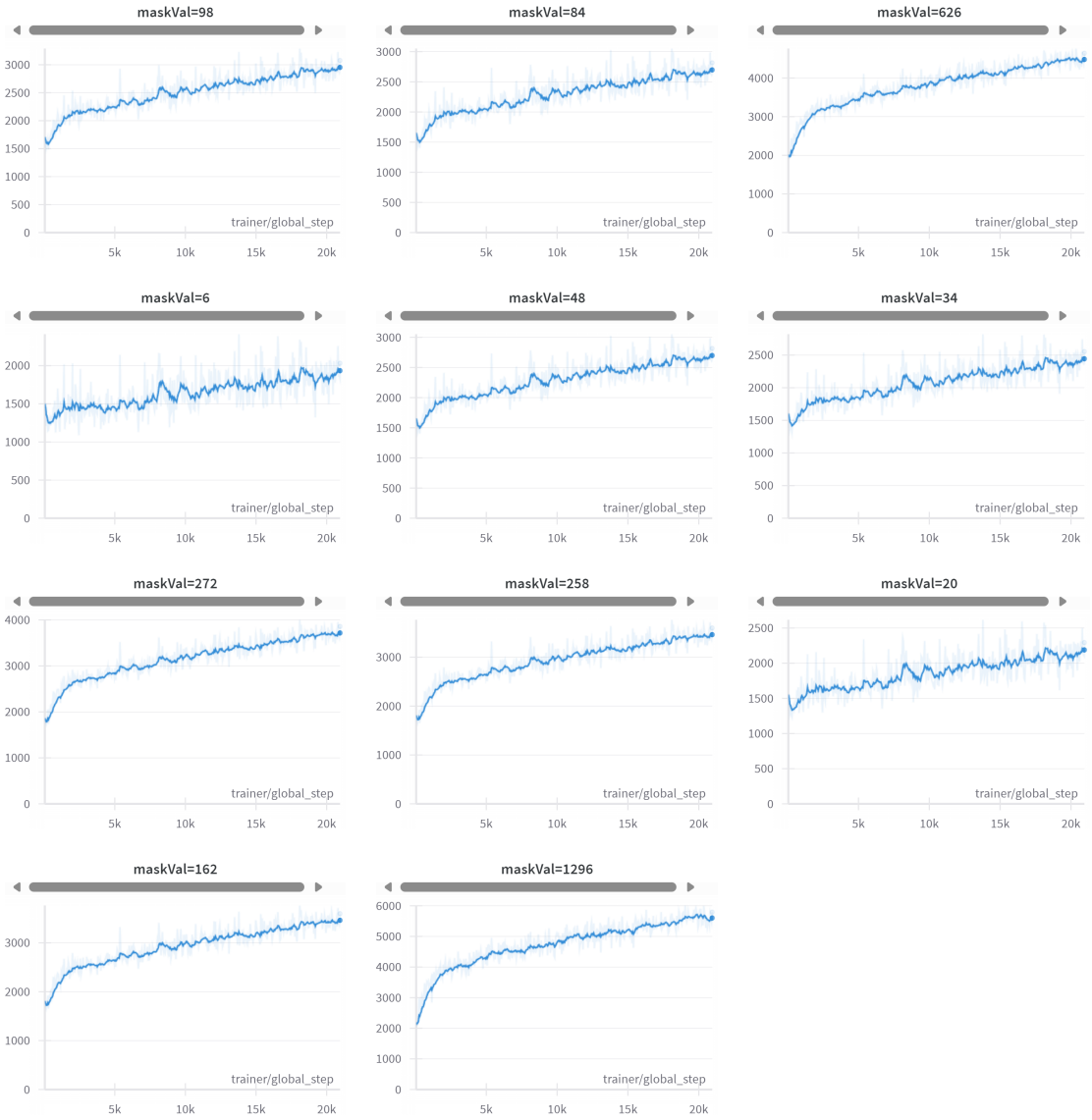


Table 5.9: A table of figures showing the loss by each region. Note the changing axis but colocated spikes

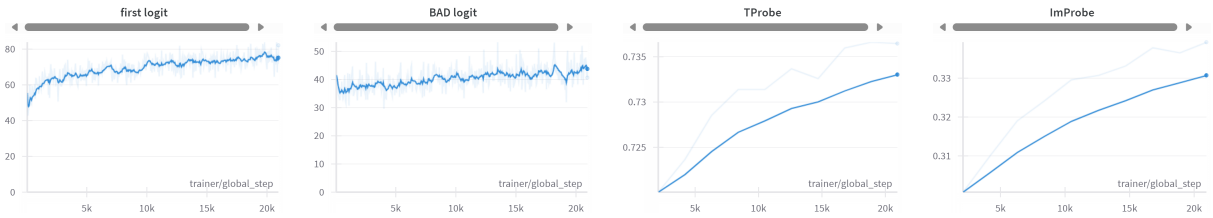


Table 5.10: A table of figures showing the training heuristics

In Table 5.10, 4 metrics are used to show that the training is going well according to the downstream applications. Firstly, the difference between good and bad logits is tracked, with the optimal logits maintaining a significant gap to the bad logits. This naturally suggests that this is a very stable training method and further suggests that we have not reached the limits of this training paradigm.

Linear probes are also monitored during training. This shows how well a logistic regression of the output can predict class. This run scores 43% zero-shot top-k = 1 on ImageNet. But the probes give a modality-specific measure throughout training. We can see from these graphs that they are relatively stable in value as a percentage, which suggests that during the validation stages, from the first epochs, reasonable embeddings are learned. It needs to be considered that there is a suboptimal maximum value to these probes by the way the ground truth is collected. We take the first class available in the instance's annotations, which is seldom representative of the image, but hopefully a semi-learnable bias that will favor images with few classes present. It also uncovers the inherent bias present in a logistic regression method that these metrics do not start from 0. For a better understanding, it may be worth looking at the implementation of these during validation to understand that they are not comparable across runs, methods, or even implementations : especially with such limited resources available. A key trade-off during the implementation here is the limited number of steps. These models were not allowed to fully converge. An approach considered was to train from the previous epoch as a starting point. Such an approach has problems such as: disadvantaging the first run; not handling large changes in embedding location, and still never fully converging.

However, these metrics all point to a successful training approach, which has occurred in a small fraction of the resources available to large models.

This training report has shown that using the config results in the best value shown in Table 5.11 and Table 5.12.

Parameter	Value
adam epsilon	0.00000001
batch size	6
dims	6
embed dim	512
exactlabels	0
JSE	0
learning rate	0.00001
logitsversion	8
logvariance	false
maskLosses	0
meanloss	false
normlogits	false
projection	”None”
prune	false
totalsteps	200,000
trainbatch size	6
transformer heads	16
transformer layers	3
transformer width	512

Table 5.11: Configuration Parameters

Metric	Value
BAD logit	40.648048400878906
epoch	10
first logit	82.13754272460938
ImProbe	0.3392668776371308
logit scale	17.2613468170166
maskVal=6	2,032.41259765625
maskVal=20	2,293.30419921875
maskVal=34	2,554.196044921875
maskVal=48	2,815.087890625
maskVal=84	2,815.087890625
maskVal=98	3,075.979736328125
maskVal=162	3,597.763916015625
maskVal=258	3,597.763427734375
maskVal=272	3,858.65576171875
maskVal=626	4,641.3310546875
maskVal=1296	5,792.8486328125
Mean Projection Value	0.00000971413828665391
mean validation stock logits	0.046441663056612015
meanloss	2,684.622314453125
proportionmaskVal=6	0.7570571899414062
proportionmaskVal=20	0.8542371988296509
proportionmaskVal=34	0.9514172673225404
proportionmaskVal=48	1.0485973358154297
proportionmaskVal=84	1.0485973358154297
proportionmaskVal=98	1.145777463912964
proportionmaskVal=162	1.3401378393173218
proportionmaskVal=258	1.340137600898743
proportionmaskVal=272	1.437317967414856
proportionmaskVal=626	1.7288581132888794
proportionmaskVal=1296	2.1577889919281006
TopK Imagenet	0.41128
TProbe	0.7364627285513361
train loss	1.5338046550750732
val loss-stock	0.8291866779327393

Table 5.12: Results Table

5.9.2 CKA Analysis

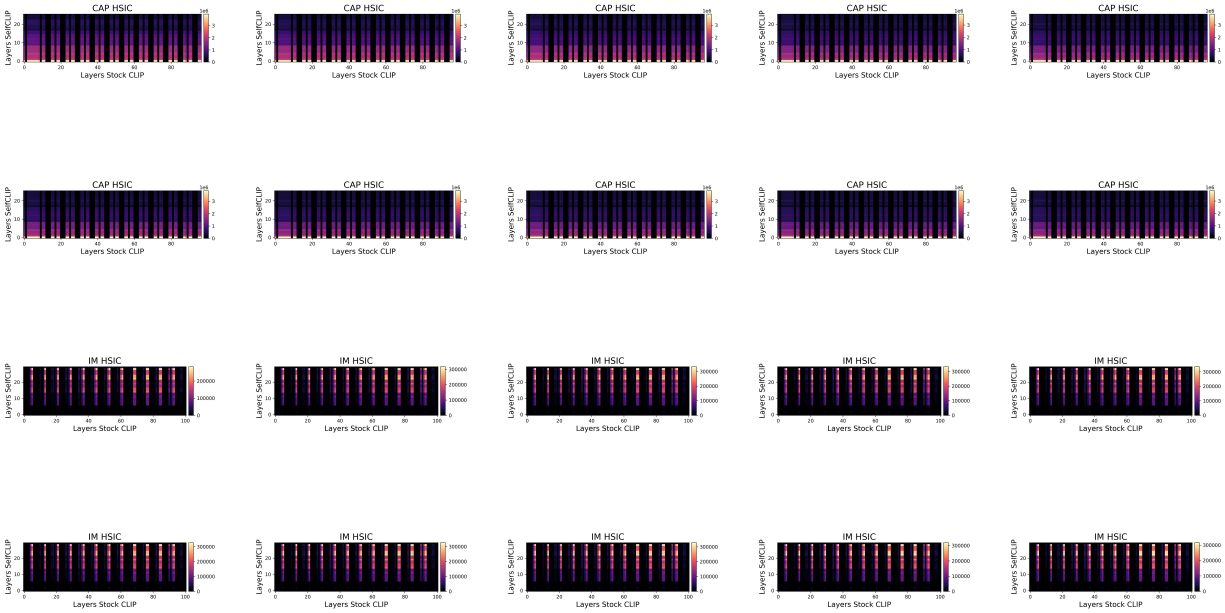


Table 5.13: CKA plots from training runs, showing the Kernel alignment of the trained model to the stock CLIP model

The CKA analysis captured throughout the training shows an interesting comparison to the stock clip model. The first 2 rows, representing the activations of the training text encoder compared to the stock implementation, show that the initial layers learn very similar values to the stock model, which deviates over training. However, the image encoder, (a different architecture), only has later layers with a similar activation. The ResNet architecture at the start of this network is unlikely to find similar activations on such a small set of images.

The first rows of Table 5.13 show how the trained text encoder compares with the stock model under CKA analysis, as discussed in Chapter 4. The diagram shows how the first layers of the text encoder learn very similar weights to those in the pre-trained model. The final layers do not. There are several plausible explanations for this phenomenon:

- Not enough training data to well train a text encoder - less than even most Low-resource languages! This is perhaps more significant when considering the imbalance of using the activation of a single token.

- The image set is comparatively limited and uniform compared to open-set data. This effect could equally be caused by overfitting to the set rather than aligning with the pretrained model
- Despite the best efforts to align the similarity metrics in the model to cosine similarity, it is highly plausible that the domain imbalance has caused a different distribution for the final textual outputs.

Compared to the pre-trained model, the image encoder appears to have a very strong correlation.

5.9.3 Comparison of Top Runs

To explore how each experiment affects training with high numbers of inputs, the best runs have been plotted, with zero-shot TopK imagenet scores of 30%. This will filter out the graphs and show the optimal behaviors in all runs.

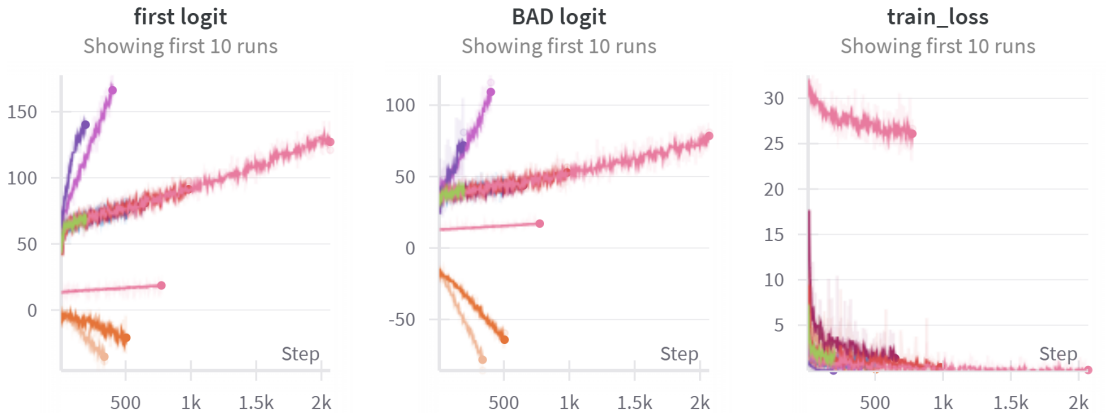


Table 5.14: A table of training metrics, showing a consistent difference between the first and bad logits. The training loss table has an outlier, with a high value, representing a run which used the sum loss rather than the mean to calculate gradients.

The plots in Table 5.14 show that in all the good cases, the positive and negative cases remain well separated, and the train loss, based on a contrastive training of the similarity logits, is very consistent across each good case. All the training loss curves exhibit similar descent parameters per step. Given such variance in hyperparameters, indicates the power of the presented methods lies primarily in using more dimensions rather than the fragility of certain hyperparameters or methods. The one outlier in the training curve plot uses region masking which, as previously discussed, separately

accumulates the loss between regions. In practice, it has a ratio between good and bad logits that is similar to several other well performing runs.

Table 5.15: A set of metrics for the top runs

The plots in Table 5.15 all show positive improvements through training: growing Image and Text probes, and improving logit-scale. The notable features of the linear layers are that the lines largely occupy the same shape. Beyond the initial epochs, the improvement and leveling off of these probes is indicative that very fast convergence is being achieved, and that linear probes are reaching their effective limit due to either the efficacy of the limited data available or the errors introduced by other paradigms. The latter is favorable as an explanation as the mean similarity score is descending, which indicates that the encodings produced are becoming more distinct and farther spaced around 512-space. This suggests that the models are significantly moving and the increased sparsity of vectors might be having an adverse effect on the ability of projections.

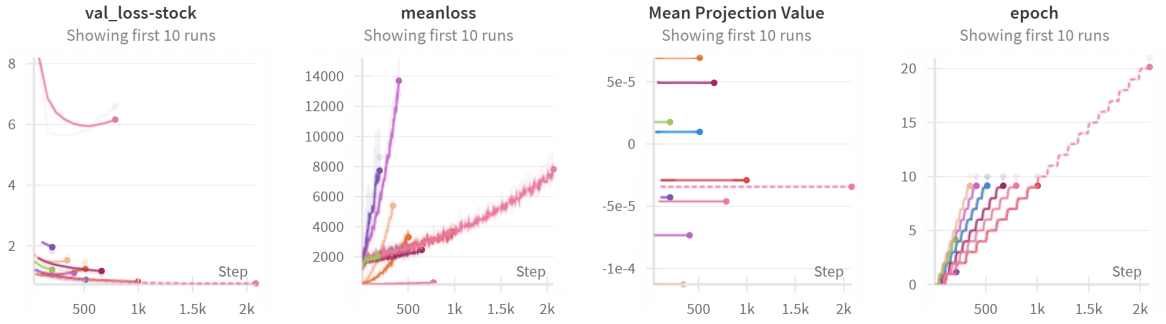


Table 5.16: A table of plots from the top 10 runs

The validation loss stock shown in Table 5.16 of these top runs suggests the encoders are not producing comparable embeddings. In spite of being high-performing

runs by both the measures of TopK and linear probes, the validation loss increases. The result seems counterintuitive until considering that compared to the original CLIP paper, the breadth of classes used does not have the same breadth; thus, samples are less likely to be well distributed in an embedding space. The observation that methods that perform well but do not have a comparable cosine similarity suggests both a lack of training at sufficient scale and reinforces the hypothesis that different similarity measures naturally produce different embedding distributions. There are a plethora of subsequent experiments that could prove this interpretation, such as observing the embeddings deviate from a pretrained model during training. However, there are insufficient resources in an academic setting to load both the trained and the pretrained models during training, compare both sets of embeddings during training, and express the trend over a meaningful training duration.

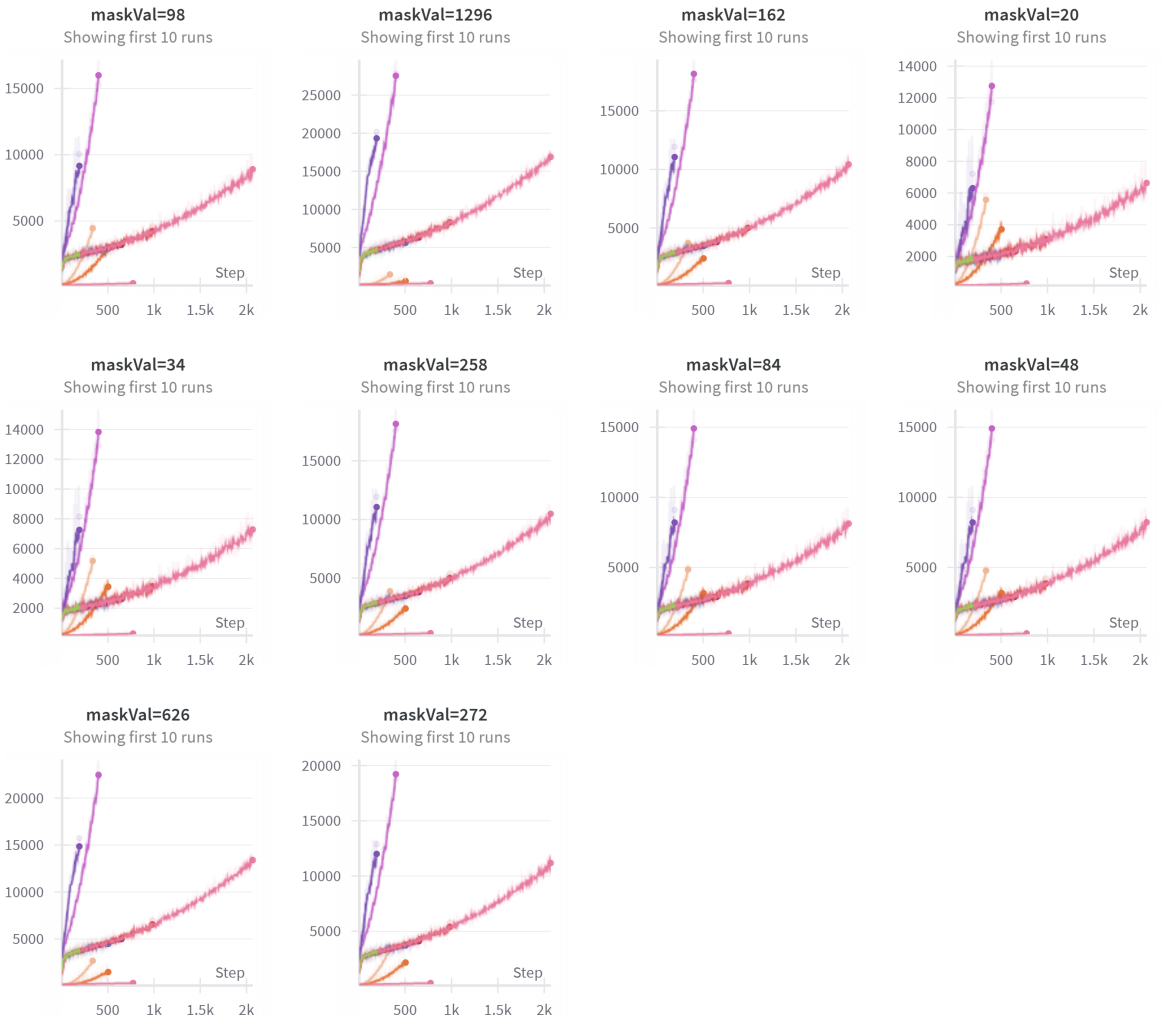


Table 5.17: The Mean Squared Error of each region in the top 10 runs.

The mask values generated during this training are interesting in comparison to the Table 5.17 because they represent the mean square error between the regions' values and the ideal labels. So they are affected by the volume of the region and the label scheme used. The differing shapes and scales of these graphs can be misleading, hence the subsequent proportion graphs. However, when interpreting these scales, they need balancing against the relative sizes of regions as discussed in the prior chapter(s). All the lines follow a similar shape, suggesting that the similarity values in each position are deviating from the optimum value somewhat. Using the variance each region might have as explored in Figure 5.13, compared with the average parameters of $n = 6, B = 8$, suggests that these summed values across whole regions reflect relatively

small deviations for individual values.

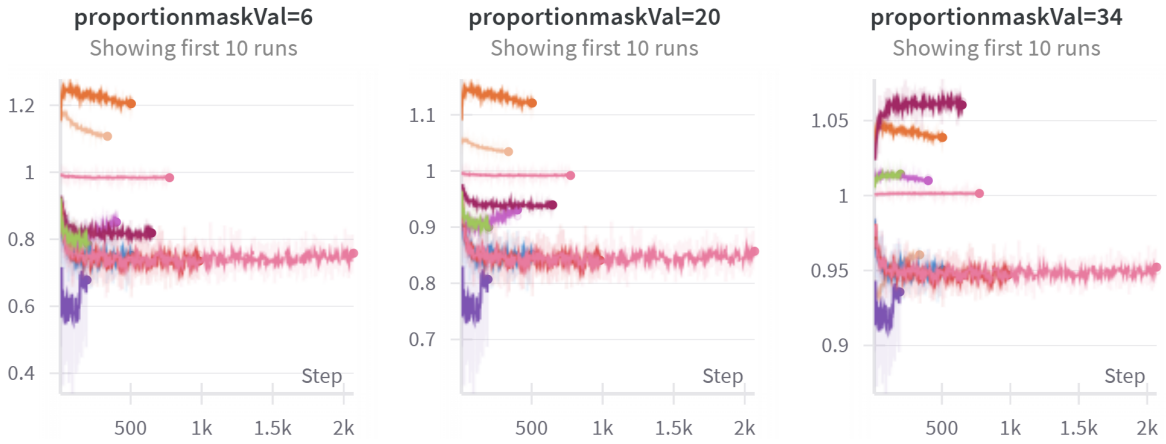


Table 5.18: The comparative losses by regions with no matches

The proportional masks reveal a lot about the training parameters of the top runs. Firstly, whereas an absolute value, the purple line is an outlier; when considered as a proportion, it is congruent to other loss metrics, indicating that the loss may be measured from a fixed point that is simply becoming increasingly far from where the vectors actually are. As discussed in the analysis section, it is very unlikely that the optimal value of 1 for cosine similarity will ever be reached. Notable in the figures for values 6-34 is that the shapes of the lines are largely the same with some flipped. Values ≤ 1 indicate that these regions contribute less than average to the overall loss. Such low index values indicate that coordinates are all unique - so this is the worst case. These being the best runs, all of them in this region diverge away from the value 1 and appear to return very quickly compared to other training runs and other works. The logit region representing all unique values is interesting for deviating the least - representing the case where poor similarity is present, and correctly so. This region is also one of the more numerously populated; thus, it does not deviate much from the perfect case. Comparatively, the regions represented by 20, where only a pair of items match, and 34, where 2 pairs match, exhibit differing behaviors: the lines are in different orders. The different orders in this case represent different hyperparameters for both labeling and logitversion. The logitversion governs how the similarity is calculated, with slightly different distributions and emphasis on pairs over large sets, and also for how the labels (and therefore distance to best case) are calculated.

The graphs in Table 5.19 each represent the cases where there are more than 3 agreeing coordinates or where there are 3 pairs (48). The order of the lines on the graphs is now congruent. This suggests that the difference between these top

functions is how a single, double, and triple pair are treated. This is perhaps most tangible in the jump between 2 and 3 pairs, where the lines reshuffle again, but all the shapes mirror around the value 1. Opposite shapes mean the systems where 2 pairs constituted a low percentage of loss are more affected by 3 pairs and vice versa.

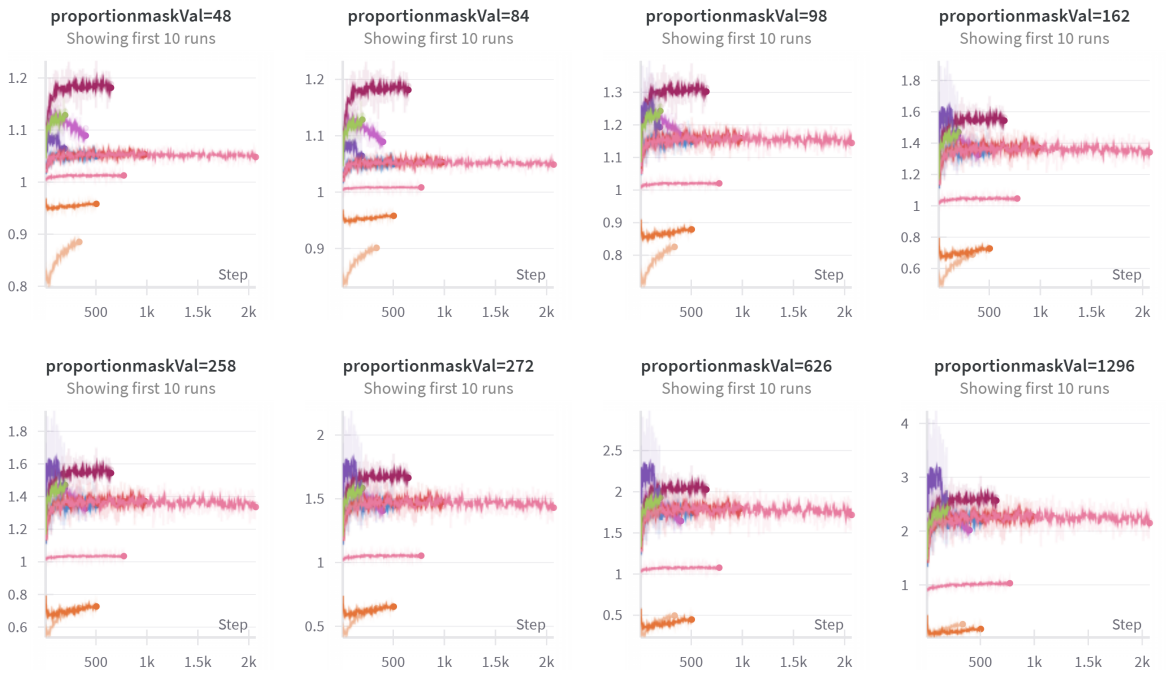


Table 5.19: The loss by region plotted as a proportion of all loss

5.10 Importance of Hyperparameters

The hypothesis being tested in this chapter is that scaling laws apply in settings where $n > 2$. To explore the many hyperparameters without thousands of runs.

To explore the importance of different options, a random forest is fitted to the hyperparameters using the many different metrics, including the Linear Probes discussed in the prior chapter. The results are shown in Figure 5.19.

Figure 5.19 shows the aggregated importance between all the metrics. The 3 most significant factors in performance are the similarity measure used, the mask of regions, and the learning rate. The latter result is a surprise given the relative insignificance of parameters like layers and parameters. However, in the test sweeps, some runs had a very high learning rate to find out whether the hypothesis that more logits allowed a better aggregate gradient was true.

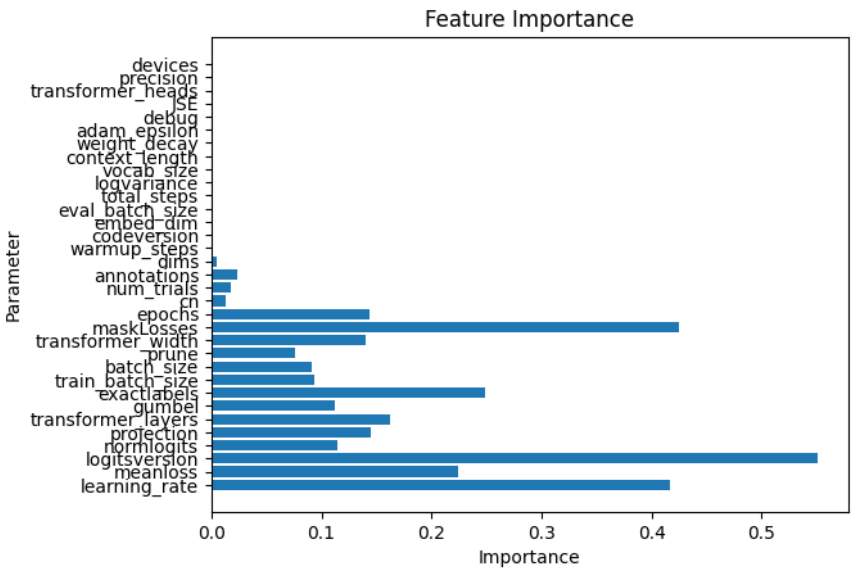


Figure 5.19: A figure showing parameter importance for $n = 6$

Using Spearman and Pearson correlation to evaluate the correlation of Learning rate yields Table 5.20, demonstrating that there is a strong negative correlation between learning rate and performance against all metrics. There is a small positive correlation between logit ratio and learning rate. A high learning rate is therefore useful in some training for quick descent, but clearly leads to high entropy and unstable representation. This is further explained by the logarithmic scale being used for LR parameters, which neither Spearman nor Pearson are particularly sensitive to (recalculating the table treating LR as categorical shows an even more extreme set of results.)

Metric	Parameter	Importance	Pearson	Spearman
Ratio between logits	LR	0.2106478	0.154691	0.092788
Image Probe	LR	0.069374	-0.214251	-0.140264
Text Probe	LR	0.148801	-0.363536	-0.214008
Ratio between logits	maskLosses	0.01924205	0.007934	0.043952
Image Probe	maskLosses	0.213682	-0.251859	-0.247412
Text Probe	maskLosses	0.198065	-0.300921	-0.303283

Table 5.20: Table showing the importance of learning rate to key metrics

5.10.1 Similarity Evaluation and Region Masking

From Figure 5.19, both the masked regions and the calculation are important. The figure is aggregated across metrics, so it may be statistically significant how similar Learning Rate and mask loss values are. Given the prior discussion that Learning Rate had some extreme, poor-performing values in the hyperparameter sweep. Mask losses must be similarly investigated given the significant impact on training.

To investigate the effect that the Regions have and the mask losses on the training with respect to these metrics, the Spearman and Pearson rankings are repeated, separating the logits version and masking criteria into categories based on a value that then holds a boolean truth value, where only a single category can be true at once. The goal is to separate out the most important approaches, showing which of the algorithms can be trained to. The parameter importance is then multiplied by the correlation (summed across both measures) to show how the parameter affects performance.

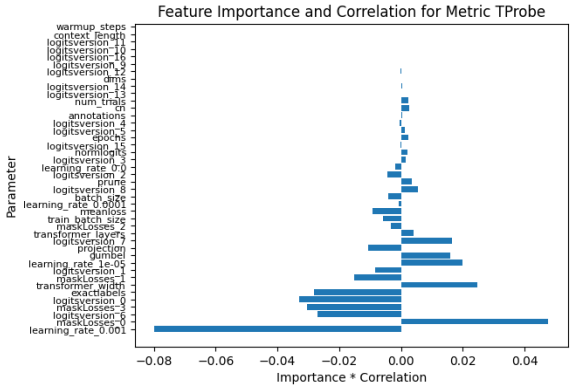


Figure 5.20: The relation of HyperParameters to the Text Probe

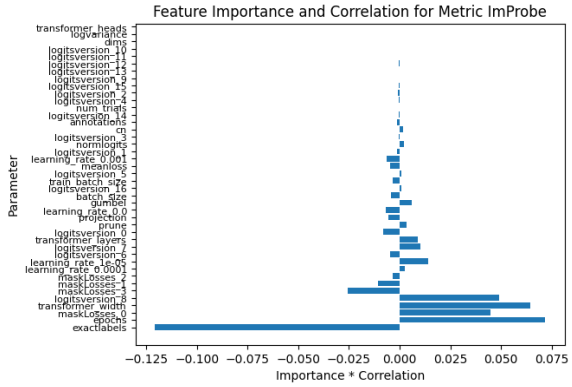


Figure 5.21: The relation of HyperParameters to the Image Probe

Figures 5.21 and 5.20 show that the Similarity metric (aka logitsversion) 8 is one of the best performing, which validates one of the peaks in Figure 5.11. To find isolate this across values other than $n = 6$, the random forest is widened across different training projects.

Evident from Figure 5.22 is that the topography, and order of importance of parameters has changed relative to the other experiments. Figure 5.22 shows that the number of dimensions is one of the key factors, reinforcing the experiments in Chapter 3, with poor performance in with $n = 3.5$, which includes re-encoding predicted text through a minimal language head. As discussed earlier, this approach lacks the modern sophistication of LLMs with tools like beamforming, thus resulting in very poor performance comparative to other experiments. Experiments with $n = 3$

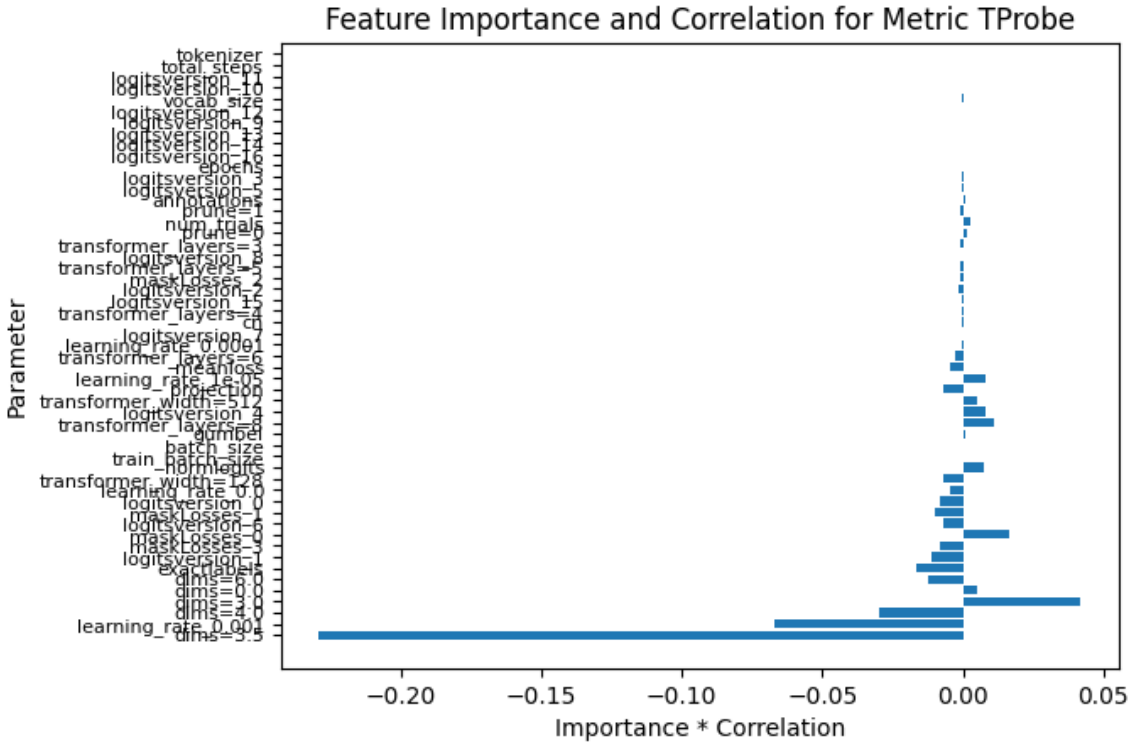


Figure 5.22: Parameter Importance for many values of n

positively impact performance, better than the stock implementation (written as $n = 0$).

This chapter's exploration into $n = 6$ has high importance but a slight negative correlation, so it is reported as a negative product. This is likely due to the distribution of logits versions that are rated as having low impact, a factor that only appears with $n = 6$.

The next ranked results, before batch size, or model width or layers, is the labeling scheme used. Using exact labels is detrimental to training. Likely due to the noise in the annotations. Further work could be to look at not only calculating across multiple batches, or at least, homogenizing the labels by region prior to training, to see whether the poorer performance is solely related to noise in the calculation, or implementation-based.

Breaking this down to isolate performance of different similarity measures, Figure 5.23 and Figure 5.24 demonstrate the comparative performance of the algorithms previously profiled.

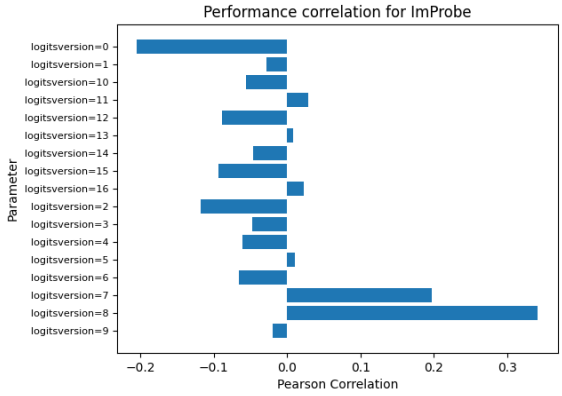


Figure 5.23: Correlation between Similarity metric and image classification performance

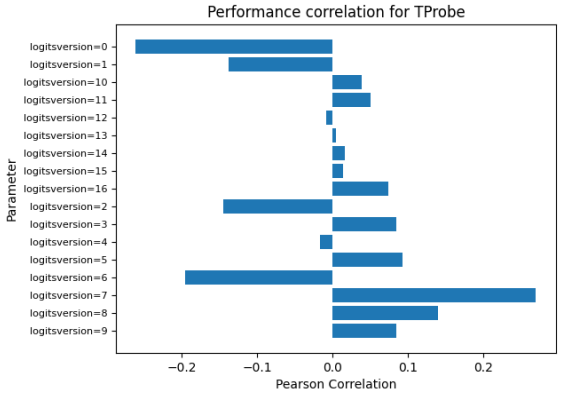


Figure 5.24: The correlation between Similarity metric and Text classification performance

The best performing methods are clearly 7 and 8. There are many factors that could cause the stronger or weaker correlations in image and text probes. From the above figures, it can be safely concluded that the best methods for $n = 6+$ dimensions are

$$S = 1 - \sum_{f=0}^{F} \sum_{i=0}^{n} (x_i^2) - \frac{\sum_{i=0}^{n} {x_i}^2}{n}$$

or

$$S = 1 - \sqrt{|\sum_{f=0}^{F}\sum_{i=0}^{n}(x_i^2) - \frac{\sum_{i=0}^{n} x_i^{\,2}}{n}|}$$

Which are similar to each other, differing solely by finding the root of the distance function that is unlikely to significantly affect the topology of the function in high dimensions. The impact on correlation between modalities typifies the difficulty with algorithm curation - where minor seemingly insignificant factors can significantly affect performance, especially on a small scale.

5.11 James-Stein Estimators

Something to consider in these multi-dimensional loss paradigms is the effect that additional dimensions have.

Assuming that two random vectors are compared, it would be reasonable to assume a normal distribution if repeated. Therefore, in effect, we have a model that works by estimating the center of that distribution and applying it to each input item. Not immediately, but applicable directly to a similarity measure, many of the proposed methods work by first calculating the mean between the values of (3+) and then the distances. If there is a better measure for where that mean should be measured, this is significant across so many samples.

The James-Stein estimator [63] approach suggests that the mean of the distribution is no longer the best approximator; instead, applying a shrinkage factor offers a better predictor of the center of each distribution.

When we consider a single sample from a normal distribution, it is most likely the mean value. Therefore, we can say that when estimating the mean of a normal distribution from a single sample, that sample is the best estimate.

However, in $n = 3+$ or more dimensions, the sample is no longer the best estimator for the mean.

While it sounds counterintuitive, James-Stein's findings show that applying a slight shrinkage factor to the sample improves accuracy. In simple terms, the volume of points that are improved towards the mean outweighs the points negatively affected by being moved away from the mean. This effect only increases in additional dimensions.

Therefore, since our method is predicated on and takes advantage of the increased volume of additional dimensions, it is worth exploring the same shrinkage factor.

```
function apply_JSE_factor(caption_features1, caption_features2, caption_features3, ↩
    caption_features4, caption_features5, image_features, gelu_function):

# Step 1: Stack all feature tensors
stacked_features = stack(caption_features1, caption_features2, caption_features3, ↩
    caption_features4, caption_features5, image_features)

# Step 2: Compute the squared norms of the stacked features
squared_norms = square(stacked_features)
```

```
# Step 3: Sum the squared norms along the appropriate dimension
summed_squared_norms = sum(squared_norms, dimension=0)

# Step 4: Compute the JSE factor
JSEFactor = 1 - (4 / summed_squared_norms)

# Step 5: Apply JSE factor to caption_features1
caption_features1_adjusted = multiply(caption_features1, JSEFactor)

# Step 6: Apply GELU activation function
caption_features1_activated = gelu_function(caption_features1_adjusted)

# Step 7: Return the adjusted and activated caption_features1
return caption_features1_activated
```

This approximation assumes that all the values in a calculation are equally weighted in their value as approximations. This explains why applying a shrinkage factor to the calculations does not improve training and is often detrimental. However, beyond $n = 4$, the behaviour of certain logits is highly predictable according to their coordinates. Further, being able to predict the variance of logits given the position in each dimension is very precise. Given that these masks have been generated for logging purposes anyway, applying varying shrinkage factors to different regions according to the number of logits in each is an approach to grow or shrink values. Although this is the same effect as scaling loss via hyperparameters.

5.11.1 Results of JSE

JSE as applied to the vector or logit output shows no discernible improvement. In some cases, it caused training instability in the initial steps of a run. This is likely due to the unpredictability of the shrinkage on negative or near-zero values.

5.11.2 Future Tests

JSE does not apply natively, as it is an approximator of the mean. It is also problematic to assume that the error is Gaussian on a given approximation of an encoder to any consideration of grounded or "true" value in a subjective space. Therefore, it needs to be applied specifically somewhere where there is a proven normal error.

5.11.2.1 Applying JSE to Specific Mean Calculation

Something to consider is how the different loss methods tried behave. In a case where there are 2-6 vectors, we evaluate the distribution of similarities between random vectors.

This will show whether JSE can be applied to our improved methodologies, showing whether the above generalisation is flawed or just the implementation. This test will show whether JSE is a transmutable operation in. (Whether it can be moved in the order of execution).

5.11.2.2 Method Discussion

This could be implemented as a test flag during training, which will dictate which mean function is served at runtime by the method factory (an architecture used to support TPUs). As many mathematical functions actually simplify away from neatly having a mean formula (as seen by the expanded formula in previous sections), therefore anything that sums all vectors is considered as a multiple of the mean and, therefore, subject to a shrinkage calculation under James-Stein estimation.

There may be further use for this method in subsequent works, though exploration of these deep statistical methods is somewhat beyond this scale of work. Notably, in retrospect, the best way to implement this is to re-sample distributions into a normal distribution with the shrinkage factor applied as a difference in mean and variance, as done with later work. This offers a cleaner computational graph, though it is still unclear whether this would affect the overall training paradigm: There are very few works that combine high-dimensional methods.

5.12 Chapter Summary

RQ.2 invites exploration of the advantages in scaling where multiple encoders are introduced. It has been well established that with the correct methodology, it is inherently possible to learn multiple modalities effectively at once using the power of n-dimensional similarity comparison. Utilising n-dimensions has a provable performance benefit, achieving impressive results in a very short time compared to state-of-the-art approaches. This chapter has answered this research question by demonstrating the effectiveness of alternative similarity measures to efficiently compare n vectors while preserving a gradient, arriving at the conclusion that the best similarity measure to use is:

$$S = 1 - \sum_{f=0}^{F} \sum_{i=0}^{n} (x_i^2) - \frac{\sum_{i=0}^{n} {x_i}^2}{n}$$

In this chapter, training has been documented using a 6-dimensional similarity comparison. Algorithms have been presented, derived from other existing methods, that extend similarity measures across multiple terms, which can aid model training. This chapter has therefore furthered the evidence that the efficacy of this methodology comes from a direct comparison of contrastive logits, and this scales with the number of contrastive logits rather than being any function of cosine similarity or transformers. Because such impressive zero-shot performance has been achieved in less than 12 hours, a minuscule fraction of the closest related work, it can be concluded that the limiting factor in reproducing these large models in an academic context is the scaling of this training and data availability.

This chapter should be used as a justification for more datasets to be produced in the form of MSCOCO with multiple candidate annotations that allow models to make full use of all human annotations.

Chapter 6

Linear Sum Assignment

The underlying message of this thesis has been to explore the grounding of concepts in a data source against multiple modalities. The exploration of grounding has been achieved by evaluating systems that exhibit multimodal activations and exhibit behaviors associated with multiple orders of knowledge.

The following chapter is dedicated to the study of what happens when training supervision is similarly challenged by addressing the balance of truth in label assignments. To answer the **RQ.4**, many of the semiotic studies that underpin **RO.1** point to grounding and assignment as fundamentals in attributing to a model's 'understanding'. What are the impacts of approximating these assignments or using noisy approximations?

In numerical computations, using limited precision can lead to inaccuracies and loss of information, especially when dealing with complex algorithms or large datasets. In this Chapter, the impact that having a finite set of values in a cost matrix will be explored for subsequent optimisations.

In relation to the wider body of work, assigning which predictions are to be learnt from, which are to be ignored, is a core epistemological function. With noisy, conflicting, and subjective annotations, and even in complicated computations of attention mechanisms, selecting optimum assignments is critical throughout ML pipelines. Tying to the prior Chapter: an attribute of contrastive training is the implicit assignment of items within a tuple. For noisy social media data and similar applications, it is unclear how well residual blocks that carry a gradient tolerate such assignments when the data is noisy. Applications of the previous chapter for web-scraping low-resource languages can be problematic due to the noise of social media and problems with it as a language resource. Linear Sum Assignment is relevant in such cases because it allows a quick selection of the most significantly weighted values in a matrix. This chapter presents several ways this can be quickly discovered and whether it can be utilized to boost or guide the gradient of a model.

6.1 Introduction to Linear Sum Assignment

The Linear Sum Assignment (LSA) problem is a combinatorial optimisation problem that deals with finding the best assignment of a set of tasks to agents in such a way that the total cost or benefit is minimised or maximised.

Given a cost matrix C, where c_{ij} represents the cost of assigning task i to agent j, the goal is to find a permutation matrix P that minimizes the objective function:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} p_{ij}$$

subject to the constraints that each task is assigned to exactly one agent, and each agent is assigned to exactly one task.

6.2 Summary of Linear Sum Assignment in Computer Vision

In a computer vision context, the annotations must be batched in spite of irregular sizes and latterly assigned against batched predictions to calculate a gradient. In simple classification and CNN-based approaches, it can be simple to predict where the annotations' corresponding result will be found in a pooled set of features, and calculate residual gradients by selecting individual components to learn. However, in recent transformer models for vision applications, the reliance on large batches, with multiple overlapping annotations and hardware pipelining means these approaches are not feasible on a batch level. The assignment of one set of values to another which may not match in size and may have a sparse assignment score across a weight matrix is a complex issue and one that does not preserve gradient for optimisation. That is, unless there is a fixed size for both annotations and predictions above a threshold.

The primary impact of LSA is on training. In object detection tasks, a common challenge is semantic segmentation: where objects are isolated in the image. During training, the model predicts bounding boxes, which must be paired to the annotated boxes. But there are problems such as differing quantities, shapes, sizes, and contents. So Linear Sum Assignment answers the question: How to decide which bounding boxes in a batch of annotations best fit the annotated data? Although it seems intuitive to pick the closest match, prioritizing both class and location becomes non-trivial when edge cases occur: conflict, duplicate candidates, or cases where no clear candidates are proposed. To find the best solution, LSA calculates the mathematical optimum case, such that all ground-truth boxes are assigned to a prediction to maximize the overall score. However, it should be noted that this is irrespective of batching, so all ground-truth annotations from B images are compared against all the predictions

from all images. This can naturally cause confusion and lead to cases where the bounding box from another image can mistakenly be assigned.

For larger boxes that may be present for multi-object relations, multi-image confusion is more prevalent as candidate predictions may all have a non-zero overlap. Hence, cosine similarity between the GT class and the output case is regularly factored in to provide some measure of clarity.

6.2.1 Acceleration and Limited Precision

As transformers become more prevalent and problem sets more complex, the linear sum assignment issue becomes important to consider, especially with the increasing move to virtual PCIE devices that may be separated by layers of hardware and potentially networks. This makes the hand-off between accelerator and CPU comparatively costly compared to processing on a CPU. This logic step occurs in the Hungarian Matcher, which means that the indices created are subsequently a crucial part of the loss calculations on the device. In the following section, the function, purpose, and potential approximations are evaluated.

When considering the impact that Linear Sum Assignment has on training, it is vital to also recognize the importance of accelerators. In the recent renaissance of ML, an increasing number of accelerators are separated across networks from associated CPUs. In practice, this means that the cost of moving data between the 2 devices is increasingly large compared to the computation costs. This means that computations like linear sum assignment that might happen at comical speed on a CPU will incur huge overhead when deployed to large-scale hardware. To this end, it is worth comparing the performance of accelerator-based methods.

6.2.2 Alternatives in Specific Applications

In this chapter, LSA is viewed as part of a holistic framework, which can often vary the requirements and present alternatives. In computer vision, for instance, LSA is used for assigning boxes to the annotations so that a comparison to ground truth can be made. In such applications, there is typically an initial gap in training between outputs and annotations. Furthermore, transformers require a fixed number of queries that map to a small, varying number of annotations. The answer to this disparity between shapes is the use of LSA to assign pairings, such that some columns may be blank, but necessary for model performance. In the desired form of relational computer vision frameworks, a relational attribute may not have a specific or meaningful bounding box, or long-distance relations may have a sufficiently large box that IoU is not a relevant measure. In such contexts, alternatives to one-to-one matching can - and should - be considered. When deployed, users may not rely on mutually exclusive or

separate relations or classes to segment their images by; in many use cases, relations may be as nebulous or specific as a difference in verbiage buried in a prompt describing 2+ entities. In a visual domain, relations, and the grounding to other modalities can be very nuanced.

This section therefore begins with an ablative study on the hypothesis that there are some applications where LSA is not the most efficient approach to assign a bounding box to a query, only matching that query's input, while mutually excluding others.

The hypothesis devised is to use query-based offsets. The transformer outputs a grid of proposals for each input query, which will then be offset and overlayed, so that the box is now projected onto a set of pixels of size

$$\text{PixelSpace} = (\text{ImageWidth} \times \text{proposals}, \text{ImageHeight} \times \text{queries})$$

.

The addition of class and box weights by simply combining the 2 losses. The theory suggests essentially offsetting bounding boxes according to the GT and predicted class, with predictions and annotations adjusted referring to a $B \times$ height and width $\times C$ grid. With the transposition, only boxes in the correct location and class will overlap. Most importantly, the one-hot selection of the best boxes occurs naturally in the Non-Maximal Suppression (NMS) step that already occurs in most computer vision pipelines. In reality, however, this subtle change has shifted from a gradient based on Loss function where $\text{Loss} = \text{Loss}_{\text{box}} + \text{loss}_{\text{class}}$ to $\text{Loss} = \text{Loss}_{\text{box}} \times \text{loss}_{\text{class}}$ which has significantly altered behaviour when $\text{Loss} = 0$. To an extent, a backwards gradient can be restored to class offset with a gumbel-softmax, to limited effect. The class steps are so much greater than IOU, projecting into the same space stops the IOU regressive convergence. Decomposing this problem into linear algebra shows that the offset would be a fixed quantity, based on image size. A regressive model cannot fit to this and would cause non-convergence by having a null gradient in sufficiently incorrect cases.

In this respect, until RL approaches advance sufficiently for a non-regressive alternative, LSA is the best approach.

6.2.3 Background: Precise Method with Branching

The idea underlying the C-implementation in the Scipy optimized version is precisely the same as the sort method; however, as it does not benefit from the ability to quickly do sorts, indexes are used to iterate over values. This means that in real terms, for most sizes of the LSA task that would reasonably come up, the time taken is around 3-6 ms with several CPU-based list comprehensions before and after.

In practice, this is fast enough that it would be largely imperceptible and not worth the time to fix compared to the minimal compute costs it accounts for. However, in a world where green computing is increasingly important, the advancement of being able to shave seemingly insignificant chunks of time off algorithms is significant, especially for such widely used algorithms in an area that consumes a significant percentage of global compute resources.

6.2.4 Background - Hungarian Matcher

The Hungarian Matcher that is behind much of the training on open-source implementations of DETR (see the HuggingFace Library [121]) and similar models relies on the idea that to calculate loss, there is a one-to-one mapping. This means that the fixed number of generated queries has to be trimmed down to the best mapping to the annotations. Once this mapping is established, the index is then used to calculate the gradients on the corresponding network components.

There are several underlying assumptions here: samples will space evenly between the number of queries. That the weightings in the matcher are a good indicator of what needs to be corrected in by the gradient.

Within the matcher, assumptions to the LSA problem are made: the bounds and approximate distribution of the values; the rough dimensions (Queries $\times$ annotations) and the performance of cost inputs. The assumptions mean that it is unlikely that during fine-tuning more than $\frac{n}{nm}$ of a column in an $n \times m$ matrix will be of high value. The fraction of the cost matrix that needs to be considered will be explored in the approximations and is a core consideration for the stability.

6.3 Metrics

6.3.1 Metrics for Evaluating Linear Sum Assignment Methods

- **Permutation Matrix:** The permutation matrix is crucial in evaluating the accuracy of assignment methods. It represents the assignment of rows to columns and should ideally be a square matrix with only one element in each row and column being 1, indicating a valid assignment. The permutation matrix can always be padded to a square, but this work recognises that contrastive loss typically occurs in a special case where the permutation matrix is a one-hot encoding of the range function.

- **Computational Load:** The computational load is an important metric because it determines the efficiency of the assignment algorithm. A lower

computational load is desirable as it indicates faster processing and reduced resource usage, and must be viewed through the lens of acceleration within a machine learning pipeline. If an algorithm can occur on an accelerator card and better preserve a residual gradient, this is to be encouraged over accuracy or CPU time.

- **Objective Score:** The objective score reflects how well the assignment method optimises the given objective function. A higher objective score indicates a better solution, making it a key metric in evaluating the effectiveness of the algorithm, but as previously discussed, subtle changes have a big impact, and this should be taken to indicate absolute error away from the Hungarian Algorithm, rather than an absolute error. A score higher than the Hungarian Algorithm suggests that rules have not been followed.

In this section, a metric is introduced for the problem of testing LSA methods: especially where applications might be tolerant to slight errors in score or follow certain distributions that make score nebulous. It a small, but pernicious possibility exists in which the permutation matrix can have equal but differing solutions. For some domains, repeated or frequent values can occur (consider the case of NMS in CV applications). While any error caused by this is going to be an insignificant fraction, methods are still assumed to take continuous values, thus making the chance of this occurrence infinitesimally small.

6.3.2 Assignment Error

In this work α denotes assignment error, which monitors the MSE between our generated Permutation matrix P and P^{gt}. In a random tensor of size x, Table 6.3 shows the assignment error against x. Let $S()$ be a sort function, with the complimentary argsort function $A()$, Y be an array of values. The assumption is that as precision reduces, the approximation holds true:

$$A(dS(Y)) = A(S(Y))$$

However, plotting the resultant permutation matrix of $A(dS(Y))$ gives several interesting error definitions. MSE as

$$\text{MSE} = \frac{(A(dS(Y)) - A(S(Y)))^2}{x^2}$$

.

6.3.3 Error Definition

LSA is considered a maximisation problem, where the largest score is to be desired. However, this is just a single way of measuring success. In this section, other metrics are introduced that consider both the permutation and cost matrix. Considering both reflects the importance of LSA in the wider role in an ML pipeline. Sum maximisation is a succinct heuristic for where score is important, but the indexes generated have more application than being reduced to a score. The boolean mask is significantly more important within ML because the lowest scores are responsible for the largest residual gradient, which is the opposite of relying on the sum of costs, where the largest costs have the biggest impact on the metric.

Values that contribute relatively little to the combined sum represent boxes that do not fit any assigned annotation. Or simply are not the best fit to a different annotation (less likely). These provide several key functions during training. First, they enforce the spatial embeddings passed through the transformer layers. Ensuring that near-embeddings are not necessarily confused or learned. In other words, it is a way to train NMS into a transformer by not allowing it to learn and correct boxes when there are better cases available.

These poor values that contribute little to LSA also represent annotations for which no good proposals are suggested. Assigning these poor values is vitally important for starting to learn the visual entity described in the annotation.

6.3.4 F1 score

For this body of work, Precision α, Recall σ and finally $F1_{LSA}$ are defined as follows, using $\cdot$ for matrices multiplication:

$$\alpha = \frac{tr(Y^T \cdot Y_{gt})}{tr((Y_{gt})^T \cdot Y_{gt})}$$

$$\sigma = \frac{tr(Y \cdot (Y_{gt})^T)}{tr(Y_{gt} \cdot (Y_{gt})^T)}$$

where tr$(\cdot)$ returns the trace of a given matrix. The difference between precision and recall (α and σ respectively) is therefore, the orientation of the input matrix. Permuting the matrix means that the precision measure, α will always have an assignment in each dimension such that all dimensions must be matched. The same is not true for the recall score, which may contain non-assigned slices.

Naturally, the difference between precision and recall correlates with the difference in the dimensions of the input matrix. A high relative difference will create a significant mismatch between precision and recall. $F1_{LSA}$ score is therefore introduced, similar to BERTScore, and built upon in the published work.

$$F1_{LSA} = \frac{2\alpha\sigma}{\alpha + \sigma}$$

This definition of $F1_{LSA}$ score seems overly complex geometrically, but serves well for approximations which can cause columns to have multiple assignments while maintaining the overall number. $F1_{LSA}$ is a good correlation with both precision and recall, which measures not only whether the best assignment rows and columns are selected but also whether the associated permutation matrix is correct.

Using this formula, the case of multiple viable options is not excessively penalized. (Approximately cancels out in square matrix, which every input can be padded to.)

Therefore, this work presents the use of $F1_{LSA}$ in the context of benchmarking LSA.

6.3.5 Scaling with Matrix Size

In a case where there are infinite numbers of values per column with reduced precision, it is reasonable to assume that the delta between the largest and the next largest value is the highest possible delta to assign. The range of values saturates the set of expressible values. Conversely, this assumption is eroded by the case where there are only a handful of values, and no such assumption can be made. Squared error is used to measure the distance between the rank of the value in a given column and the importance of the assignment based on the ranking of the delta to the value on either side. The subsequent graphs show n regions. Each region of colour shows the rank of $S(Y)_n - S(Y)_{n+1}$. In a perfect case, each region is expected to have a sharp, uniquely situated peak density. The following table shows that the distribution of Y and the precision are significant factors.

Notably, these graphs plot whether the rows naturally fall in the correct, logarithmic scale. In practice, this is only the correct picture in the initial stages of the assignment problem.

To assume that the optimal random distribution is to have a sparse matrix X, as defined by $X\ |N(0, 0.3)|$, with which FP8 is shown to consistently outperform other precisions at all matrix sizes for the assignment of linear sums.

Each point represents the aggregate of methods and approaches. The plot shows that the drop-off when there is an imbalance is very high, unlike in square matrices. The change in performance based on the ratio of sizes is because in a square matrix, the permutation matrix is more forcing of a final assignment. The significant change in the graph according to dimension imbalance also shows that methods' performance is enhanced by the final step of finding the maximum value in the row or column. As the size of the slice relative to other dimensions changes, the number of assignments drops; therefore, the error increases. The performance boost in the minimum dimensions is

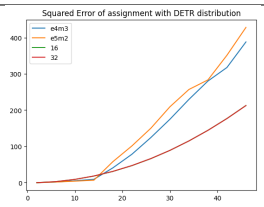
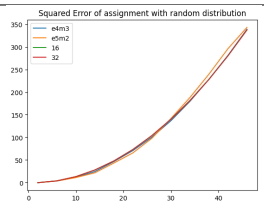
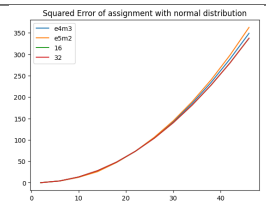
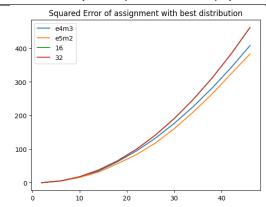
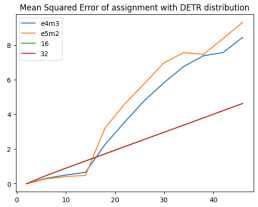
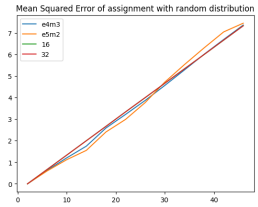
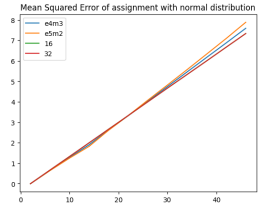
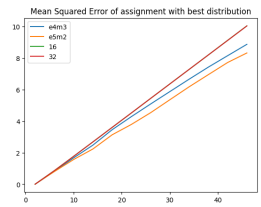
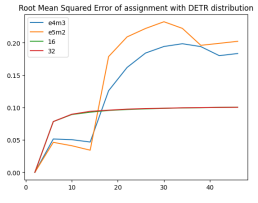
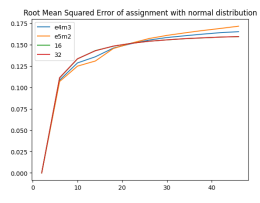
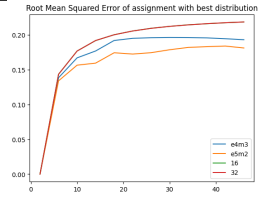
	DETR	Uniform	Normal	$X \ \lvert N(0, 0.25) \rvert$
Squared Error				
Mean Squared Error				
Root Mean Squared Error				

Table 6.1: A demonstration of how approximation error varies by number of columns in different precisions

because the probability that the maximum value in a slice being the assigned value becomes very high, and the chance of the highest value occurring in the same position in other slices diminishes.

Therefore, if a Cost Matrix C is defined as being of size W, H, then it can be said that the performance of argmax-based methods is at least proportional to $W^2 : H^2$.

It is therefore worth highlighting the importance of padding the matrices to a square for these methods when deployed, even though significant padding allows permutation errors to be included in the evaluation of methods. This is the prevalent sentiment for why assessing score is such a key metric when considering the assignments in other empirical methods, especially where dynamic sizing occurs in a specific domain, such as in a CV context where a fixed number of candidates applies to an unknown size of annotations. Naturally, the answer when using NN approaches is to train on sparse data and pad to a maximum annotation size during deployment.

6.4 LSA Improvements

6.4.1 Computer Vision Specific Optimisations

In a computer vision context, where LSA is used to assign bounding boxes, there are some specific optimisations that can be made. These optimisations come with some extra considerations from the placement of LSA in the whole model pipeline. LSA usually occurs with the same steps as Non-maximal suppression and is performed on an array separated from the computational graph. A demonstrable increase in speed can be seen with GPU-based implementations and an enabled graph using the following improvements:

6.4.2 Approximation with Batch

The Hungarian Matcher is the method beneath the implementations in the 'scipy' library. It is called on slices of a cost matrix formed with predictions in one dimension and reference annotations in the other. Each slice represents the subset of annotations for each image. Slicing like this requires detaching from the gradient and many successive calls. An improvement to this order of execution is to apply the linear sum assignment and then take the slice. There are 2 ways this can be performed: Batch or image masking.

Image masking is the approach with the best fidelity to the stock implementation, providing all the logic across the whole batch of logits by modifying the Hungarian algorithm to only mask out the part of each row that corresponds to the image of

the assigned value. It also marks a distinct alteration to the logic of a permutation matrix as values can have a repeated assignment in a row, outside specific ranges.

The alternative approach is to use batch masking. The merit of this is to allow in-batch negatives to affect assignment where annotations from other items within the batch can compete for the correct location.

The hypothesis here is that, in allowing queries to be misassigned in the case of near annotations, which queries tend to attract each class may well differ. Given that queries in DETR are typically generated by a mesh of spatial embeddings, this may be important to help break the spatial bias.

The final improvement can be seen in these 2 examples of pseudo code.

```
for each i, c in enumerate(split(C, sizes, dimension=-1)):
    indices[i] = linear_sum_assignment(c, maximize=False)
```

This code gets replaced by the tensor operations as follows where cdf and icdf are distribution functions to resample the experimental normal distribution as a normal distribution centred around 0, and C is inverted to make the LSA a maximization task rather than minimisation.

```
# Step 1: Calculate the probability using the cumulative distribution function (CDF)
prob = m.cdf(-C)  # Use negative sign so that minimize LSA becomes maximize

# Step 2: Apply inverse CDF (ICDF) to probabilities and permute dimensions
fx = inverse_cdf(tgt, prob).permute(0, 2, 1)

# Step 3: Select specific elements using diagonal lookups
fx = fx[diagonal_matrix(ones_like(sizes, dtype=bool, device=C.device))
        .repeat_interleave(sizes, dimension=1)].transpose()

# Step 4: Create row lookups using diagonal matrix and repeat interleave for lookups
row_lookups = diagonal_matrix(ones_like(sizes, dtype=bool, device=sizes.device))
              .repeat_interleave(sizes, dimension=0)
              .repeat_interleave(sizes, dimension=1)

# Step 5: Apply custom batch approximation for LSA
outputs = Batch_MyApproxLSA(fx, row_lookups=row_lookups)

# Step 6: Extract indices where non-zero outputs occur, split based on sizes
indices = [non_zero(o, as_tuple=True) for o in split(outputs, sizes, dimension=1)]

# Step 7: Return the indices
return indices


function Batch_MyApproxLSA(Batched_TruthTensor, row_lookups, maximize=True, lookahead=2):

    # Step 1: Initialize mask and results tensors
    mask = ones(shape_of(Batched_TruthTensor), device=device_of(Batched_TruthTensor), dtype=bool↩
        )
    results = zeros(shape_of(Batched_TruthTensor), device=device_of(Batched_TruthTensor), dtype=↩
        bool)

    # Step 2: Iterate through the number of columns in row_lookups
    for each iteration in range(number_of_rows(row_lookups)):

        # Step 3: Find the column with the highest value considering the mask
        col_index = argmax(argmax(where(mask, Batched_TruthTensor, 0), dimension=0))

        # Step 4: Find the row corresponding to the maximum value in the selected column
        row_index = argmax(Batched_TruthTensor[:, col_index], dimension=0)

        # Step 5: Update the mask to exclude the selected column and associated rows
        mask[:, col_index] = False
        mask[row_index, row_lookups[col_index]] = False

        # Step 6: Mark the selected (row, column) pair in the results
        results[row_index, col_index] = True
```

```
    # Step 7: Return the final results tensor
    return results
```

6.4.3 General Approximations

In this section different approximations are evaluated. As established previously, the score is less important than the rules that govern the assignment. In many cases, the largest contribution to the score is the least indicative of the best assignment. These methods are in need of revisiting after the subsequent applications are understood. Each application has marginally different requirements, and some will require more stringent rules than others, so it is worth catching the nuance of how each approach promotes different assignment geometries.

6.4.3.1 Reimplementation for Accelerators

Within this body of work, accelerator implementations have been written to approximate LSA. Accelerators, typically mirroring a GPU architecture, allow rapid calculation of simple arithmetic procedures. In recent years, approaches have been trialled to significantly reduce the precision of neural networks. Such approaches regularly use Half (16-bit) precision or even 8-bit floating point precision (FP8). Such approaches exist to maximise the amount of parameters used or reduce the memory footprint. It can be shown that FP8 does introduce error to the general case, but approximations can be steadily improved if the distribution of matrix values is resampled to a half-normal distribution.

6.4.3.2 Recursive Approximation Enabling Descent

Another way to think of the LSA problem is not as an assignment, but rather as a transformation. Taking a set of continuous values and quantizing them so that every row and column sums to 1. Where assigning each column and row has a fixed number of steps (for most ML tasks equivalent to the number of annotations), the merit of applying a gradiented function is that the number of steps may be fewer than the number of rows. Aside from the obvious speed improvement of potentially fewer iterations, the trade-off between time (or number of steps) and accuracy is now linear. Not to mention that the costly indices collation stage is much faster.

The method used to generate Table 6.1 is to subtract from every value the cost of picking it. Algorithmically, this looks like for each location (row and column) subtracting the sum of maximum values of that row and column not including itself. But this is done using array operations to minimize branching and loops.

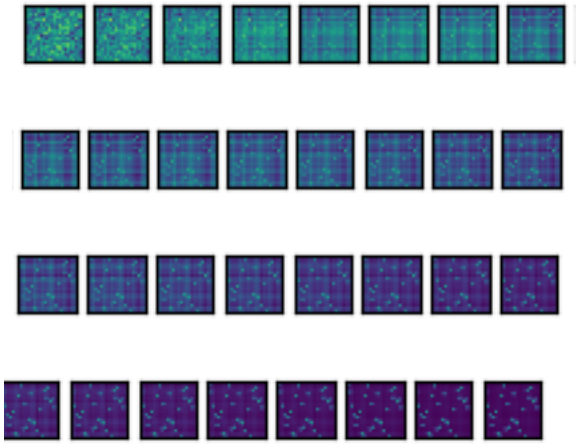


Figure 6.1: A figure showing the iterative steps of recursive Linear Sum assignment. Subtracting from every value the mean of the next highest in the row and column and repeating results in a near-one-hot solution. Top Left is the starting Cost matrix, stepping left to right.

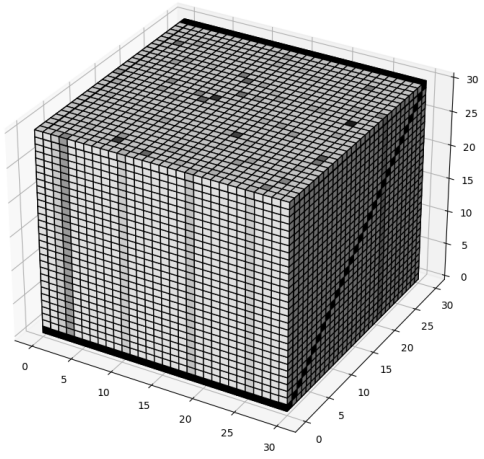
6.4.4 Method 1 - Dimensional Extension

Method 1 repeats our cost matrix, C by size n. The diagonal of n is masked, and n' taking the argmax of the result has a shape of $n' \times m$, which represents the argmax if every column corresponds to ignoring each value in turn when considering the argmax.

In Figure 6.2a the variable 'remove' is a mask, that accounts for the black diagonal in the figure. This is an efficient way to reuse this mask many times at the expense of memory complexity. However, this figure is repeated diagonally in both directions, so that the resultant argmax direction results in the sum of the next largest value in the row and columns.

6.4.5 Method 2 - Indexed Insertion

Method 2 is to realise that argmax has a very defined behavior, meaning that the argmax of the remainder of the column will be the same as the argmax of the column except for all values except the cases where it is instead the next largest value. The pseudo code for this is listed in Figure 6.3, showing how the iterating step works to slowly disambiguate values.



(a) A figure showing how the LSA matrix is expanded

```
function calculateTotalCost(rewards, ↩
    remove, cost_neg):

# Step 1: Expand the 'rewards' tensor ↩
    along a new last dimension by ↩
    repeating the last dimension ↩
    values
weights = expand_tensor(rewards, ↩
    repeat_along_last_dim=rewards.↩
    shape[-1])

# Step 2: Apply masking to replace ↩
    certain values (based on the '↩
    remove' mask) with 'cost_neg'
weights = mask_tensor(weights, mask=↩
    remove, fill_value=cost_neg)

# Step 3: Calculate the next highest ↩
    values along dimension 1 (rows)
Costs = next_highest_fn(weights, ↩
    dimension=1).values

# Step 4: Repeat the process for ↩
    another weights tensor, permuting ↩
    the 'remove' mask
weights2 = expand_tensor(rewards, ↩
    repeat_along_last_dim=rewards.↩
    shape[-1])
weights2 = mask_tensor(weights2, mask=↩
    permute(remove, order=[1, 0, 2]), ↩
    fill_value=cost_neg)

# Step 5: Calculate the next highest ↩
    values along dimension 0 (columns)
Costs2 = next_highest_fn(weights2, ↩
    dimension=0).values

# Step 6: Compute the total cost by ↩
    adding Costs and the transpose of ↩
    Costs2
Cost_total = add_tensors(Costs, ↩
    transpose(Costs2))

# Step 7: Return the total cost
return Cost_total
```

(b) Pseudo code for applying recursive LSA

Figure 6.2: The construction of recursive LSA and intermediate state

6.4.5.1 Optimal Step Count

The optimum number of steps to repeat over a matrix to get a one-hot permutation matrix varies. There is a trade-off between ensuring correctness while minimizing the number of steps taken. In Figure 6.4, no change can be seen in the latter steps. This indicates that significant savings can be found by reducing the number of steps. However, too few steps risk inaccuracy and conflicts.

```
%
function reduceLinearSumAssignment_v2(rewards, maximize=False):

    # Step 1: Find the top 2 values and their indices along dimension 1 (rows)
    Topv, topi = top_k(rewards, k=2, dimension=1, largest=maximize)

    # Step 2: Create a cost matrix initialized with the top 1 values repeated across columns
    costs = repeat(Topv[:, 0].unsqueeze(1), times=rewards.shape[-1], dimension=1)

    # Step 3: Create a one-hot matrix indicating the top 1 positions
    one_hot = create_zeros_like(rewards, dtype=bool)
    one_hot = scatter(one_hot, indices=topi[:, 0].unsqueeze(1), value=1, dimension=1)

    # Step 4: Replace the values in the one-hot positions with the second top value from `Topv`
    costs[one_hot] = Topv[:, 1]

    # Step 5: Find the top 2 values and their indices along dimension 0 (columns)
    topv2, topi2 = top_k(rewards, k=2, dimension=0, largest=maximize)

    # Step 6: Create another cost matrix initialized with the top 1 values repeated across rows
    costs2 = repeat(topv2[0].unsqueeze(0), times=rewards.shape[0], dimension=0)

    # Step 7: Create a one-hot matrix indicating the top 1 positions along the columns
    one_hot2 = create_zeros_like(rewards, dtype=bool)
    one_hot2 = scatter(one_hot2, indices=topi2[0].unsqueeze(0), value=1, dimension=0)

    # Step 8: Replace the values in the one-hot positions with the second top value from `topv2`
    costs2[one_hot2] = topv2[1]

    # Step 9: Compute the total cost by adding the two cost matrices
    Cost_total = costs2 + costs

    # Step 10: Return the total cost
    return Cost_total
```

Figure 6.3: Main function for Recursive Method 2

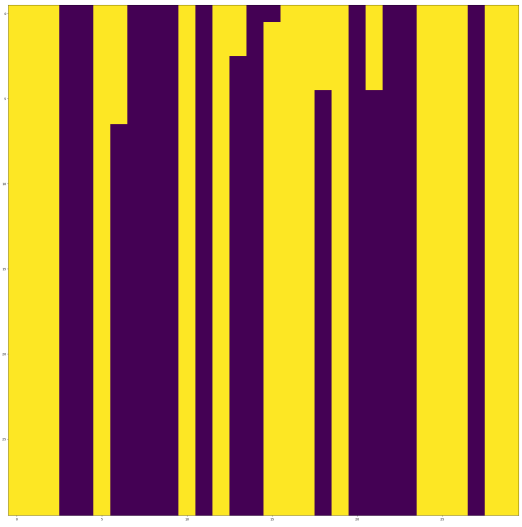


Figure 6.4: A plot of correct (Purple) vs negative (Yellow) assignment per step

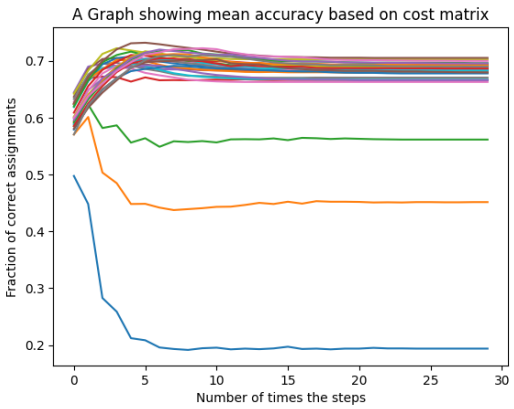


Figure 6.5: A plot of accuracy against step count for differing step magnitudes

6.4.6 Method 3 - Probablistic Approach with Gumbel Softmax

The third method presented in this section is a different view of LSA - to treat it as a quantisation problem. Quantisation problems typify the challenge of taking a set of continuous values and converting them to a one-hot selection. Describing the problem this way is very similar to the discussion in Chapter 3 surrounding the challenges of one-hot selection of vocabulary when encoders and decoders are directly appended. The Gumbel Softmax function - originally for predicting peak values in a series, produces a one-hot vector that maintains a gradient based on other values in the output. It is therefore a probabilistic quantisation function: not always returning the top value, but the error corresponds to the imbalance.

The aim is to take a Gumbel Softmax of the original distribution. Repeating this step until there are no assignment collisions.

It can be considered that the probabilistic nature may cause a significant reduction in accuracy and determinism. However, the hypothesis goes that this will introduce noise to training, allowing near misses to be trained, and should not affect well-fit models.

The advantage of this is that in the cases of well-fit models, the assignment will be as quick as a Gumbel softmax.

```
function GumbelSoftmaxLinearSumAssignment(rewards, maximize=False, factor=1):

    # Step 1: Apply Gumbel-Softmax to generate initial assignment
    initial_assignment = gumbel_softmax(rewards, dimension=0)

    # Step 2: Identify collisions (multiple assignments to a single row)
    collisions = sum(initial_assignment, dimension=1) > 1

    # Step 3: Resolve collisions by re-applying Gumbel-Softmax to collided columns
    initial_assignment[:, collisions] = gumbel_softmax(initial_assignment[:, collisions], ↩
        dimension=1)

    # Step 4: Identify free columns and rows (where no assignments were made)
    free_columns = sum(initial_assignment, dimension=0) == 0
    free_rows = sum(initial_assignment, dimension=1) == 0

    # Step 5: Reassign free rows and columns using Gumbel-Softmax
    initial_assignment[free_columns, free_rows] = gumbel_softmax(initial_assignment[free_columns↩
        , free_rows], dimension=0)

    # Step 6: Return the indices of the non-zero elements (assignments)
    return non_zero_indices(initial_assignment, as_tuple=True)
```

Figure 6.6: Psuedo code for using Gumbel softmax in Linear sum assignment

6.4.6.1 Conflicts of Argmax

For dimensions of size 1, LSA is equivalent to argmax or argmin. This equivalence breaks with larger sizes, though argmax will always be a reasonable approximation

where 1 dimension is vastly larger than the other. For this reason, in the work in the Appendix for an investigation into reduced precision, the effect of matrix dimensions on $F1_{LSA}$ score is considered.

6.4.7 Method 4 - NN Approximations of Linear Sum Assignment

In the previous methods, LSA was shown in conjunction with methods like BERTscore to reduce the double counting of tokens. Rather than directly implementing a permutation matrix as loss, it is worth exploring whether LSA can be learned intrinsically, allowing a neural network to learn the problem and be later implemented akin to BERTscore to provide a gradient naturally. Separate training has been split from other related works to allow for an empirical review of the method rather than relying on the domain assumptions of prior parts.

6.4.7.1 Experiment Definition

Evaluation of the use of LSA in training presents key challenges. First, there is a finite and discrete set of permutations that the assignment matrix (the one-hot permutation matrix, P) can take. The discretised values mean that this may have to be applied as a constant factor to a learning rate rather than directly contributing to a gradient calculation.

> Hypothesis: The introduction of a permutation matrix to contrastive training can improve the initial stages of contrastive loss training.

6.4.7.2 Method

To test whether a Permutation matrix, P, can be learnt and introduced as a parameter in a contrastive training framework, the learnability of P must be assessed with simple metrics.

An experiment is devised to compare the previously discussed methods against a Linear model and a transformer, with the assumption that the attention mechanism in a transformer may provide a means of masking the performing analysis across the bounds of the permutation matrix.

The transformer used has 3 layers, and a 512 width. Each was trained for 100 epochs, 20000 steps, using random inputs and the ground truth. The loss scales by the largest loop and all values that have significant costs associated with them, reduced.

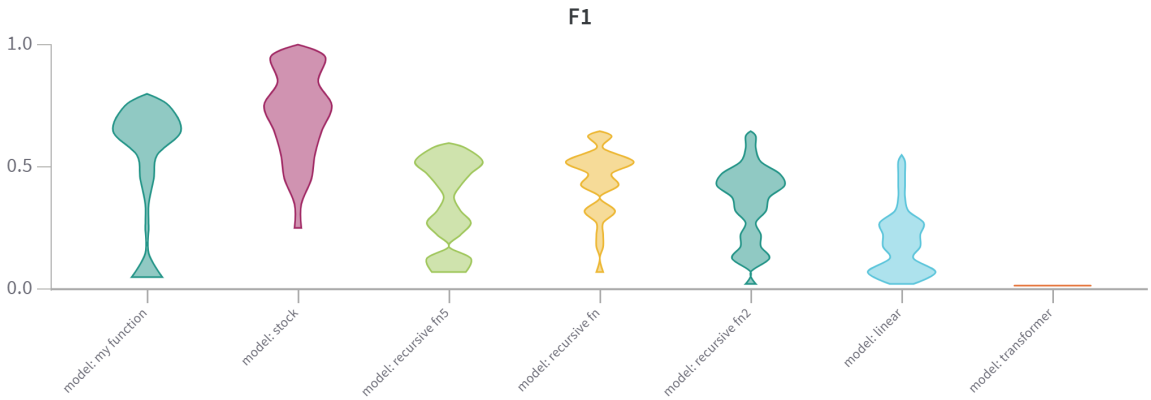


Figure 6.7: F1 score by LSA Approach

6.4.7.3 Training Results

Using just a number of linear layers with a transpose, the SOTA results are achieved.

In Figure 6.7, the performance of different approaches is compared. The approaches are:

- 'my function', a bespoke implementation assigning the highest delta between top values in columns or rows.
- 'stock', the sklearn implementation of the Hungarian algorithm implemented in C.
- Recursive family, these all have the same primary function, which emphasises the distance between peak and other values in the rows and columns. What sets these apart is the usage of different final selection methods, ranging from a matrix top-k which fails strict LSA checks, argmax, and stepped argmax.
- Linear and transformer approaches - these are trialled as the more common ML architectures - linear because the activation function can approximate the value boosting performed by the recursive function, and the transformer for the assumption that an attention layer can be more discerning in finding the peak values in the rows relative to columns

From Figure 6.7, the stock column is more revealing of the F1 score. Comparing this method with itself, the expectation would be a perfect F1 score each time. However, the scaling of width and height, and therefore empty columns in the F1 score, causes the decay in the stock method.

Once this is understood, the performance of the bespoke my function method appears to be far closer to the stock implementation by the F1 score. However, it is possible to construct cases where the novel implementation fails. This case would be where the gradient between the smallest values is steeper, which would be the case if the largest values were very close together. Thus, a strong emphasis on the experimental distribution for the cases where this is to be applied because it ought to be close to a Half-Normal distribution for optimal performance, which should be true throughout all stages of model training.

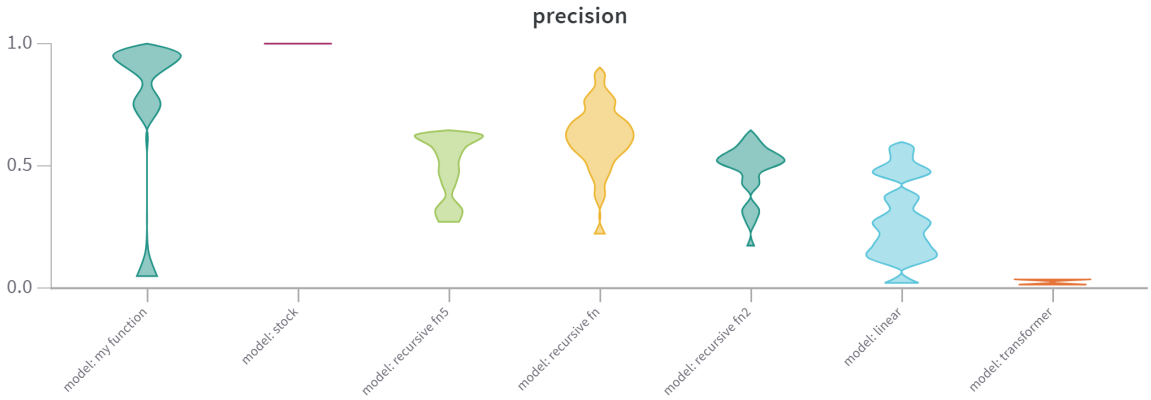


Figure 6.8: Precision of algorithms in FP32

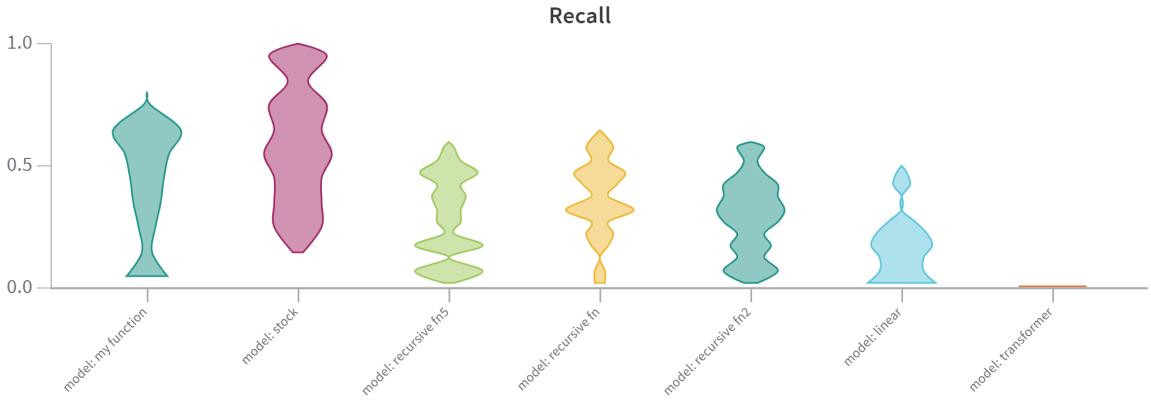


Figure 6.9: Recall of Algorithms in FP32

Figures 6.8 and 6.9 show that the previous statement on the F1 score breakdown in stock models is correct, and notably, the other methods have much more

consistent profiles across both dimensions, indicating that candidate selection is incorrect even if the comparative scores are close. Significantly, in the precision measure, the implementation of the function on this work scores very near-perfect performance, which, considering that many other GPU-based operators are equally non-deterministic, is a significant result.

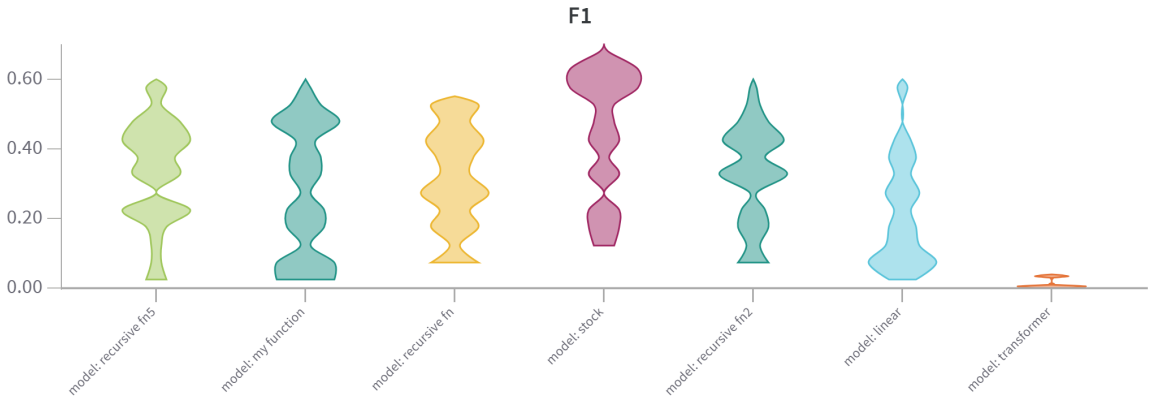
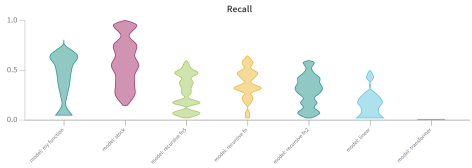


Figure 6.10: F1 Score across algorithms in FP8-E5M2

Figure 6.10 shows how the $F1_{LSA}$ is affected by reducing the precision. As documented in the paper in the appendix, as precision is reduced, a special case exists where values fall on a Half-normal distribution naturally, which radically improves the performance of approximations.

At the time of the experiments, FP8 is not widely implemented for PyTorch, though reducing the precision further to FP4 is implemented for some models. Practical limitations existed for these experiments, meaning that the implementation casts values to low precision but the mathematics occur in higher precision, meaning that the recursive algorithms are inaccurate but still perform well. The stock benchmark is reduced because the comparison is made with the actual value rather than the LSA score on the quantised values. See the appendix for more details on this work.

(a) Recall across algorithms in FP8-E5M2



(b) Precision across algorithms in FP8-E5M2

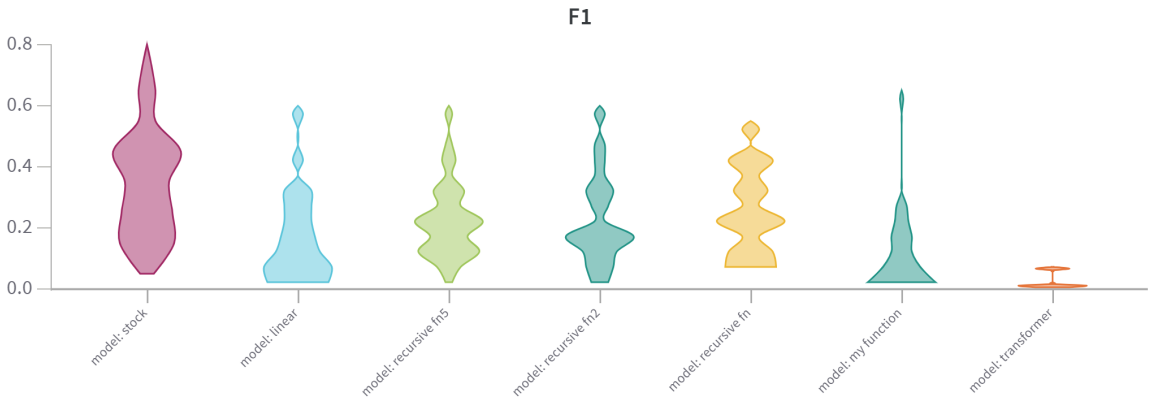


Figure 6.12: F1 Score across algorithms in FP8-E4M3

FP8 precision comes with 2 flavours as previously discussed, which affects the range and accuracy of expressible values as previously discussed. The graph in Figure 6.12 shows that e4m3 is less effective at creating the correct distribution of values compared to e5m2 shown in Figure 6.10. This figure contributes greatly to the hypothesis that the value distribution is more key to approximations than precision or algorithm.

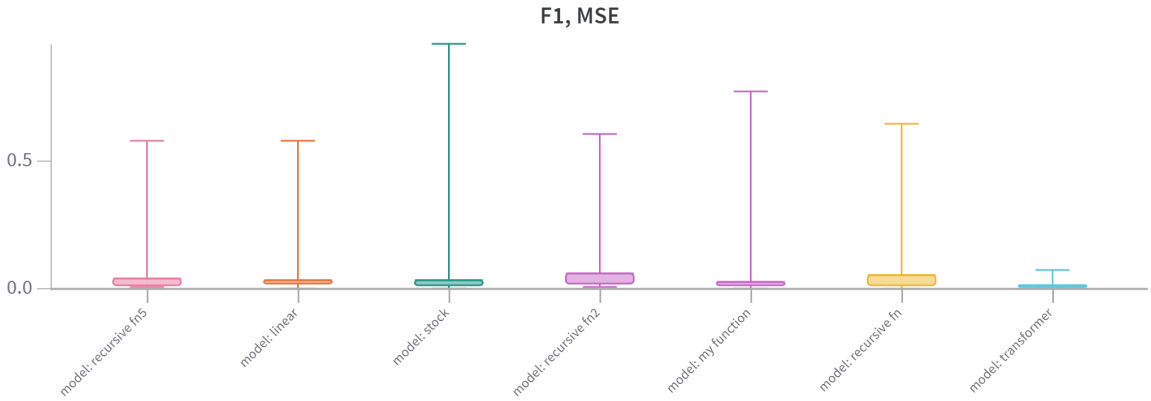


Figure 6.13: F1 compared to MSE loss during LSA training

So far, the focus has largely been on F1 score: Firstly, because it can better handle methods that deviate from the rules of LSA, but secondly because in the context of an ML pipeline, the assignments of the poor results are arguably as meaningful if not more so than the others, especially when training uses gradient methods rather than reinforcement learning. Therefore, Figure 6.13 is used to show how even though the F1 scores are high, the MSE loss comparing the input matrix with the assignment

matrix still produces a very small loss: F1 is significantly more sensitive than MSE as a metric for assignment scores.

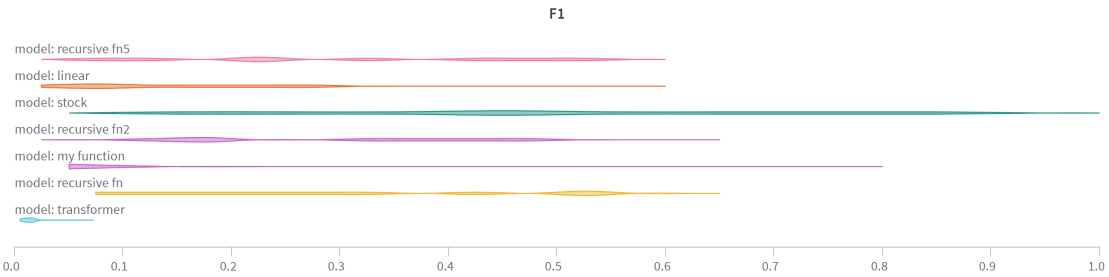


Figure 6.14: Distribution of F1 scores by model

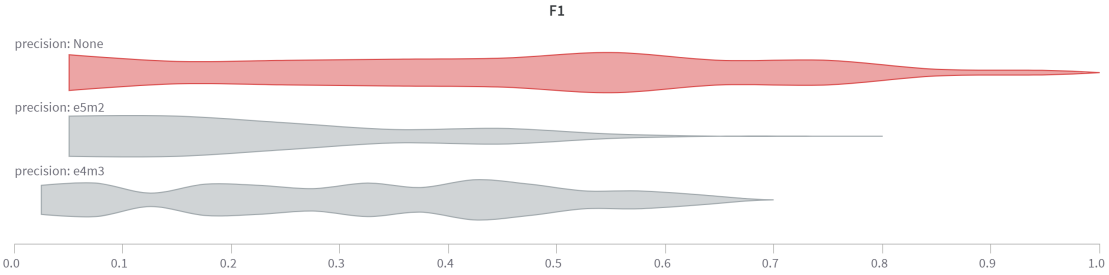


Figure 6.15: Comparison of No reduction in precision to FP8

Figures 6.15 and 6.14 show a side-by-side comparison for all the runs of F1 scores grouped only by model and precision: the resulting graph range is the most important and the distribution within it. Notably, the accelerator-based method is close to stock implementation and can perform even closer in reduced precision, as detailed in the Appendix. At the start of this section, subperfect performance for F1 scores was

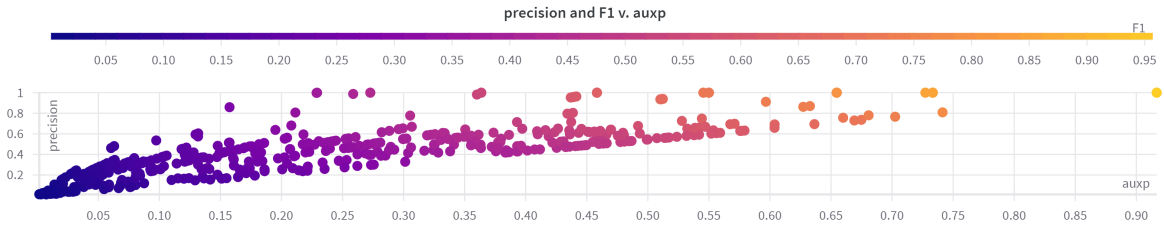


Figure 6.16: Relation between PRF

explained with the calculation of the F1 score being affected by imbalanced matrix

dimensions. The plot of all recall, precision, and F1 scores (PRF) runs is shown in the graph in Figure 6.16. In particular, the points are aligned in multiple lines, each passing through the origin. These represent the imbalance of the matrix dimensions; it can be seen reasonably easily here how the change in this ratio correlates to the distribution of points, which informs the requirement of experimental distribution listed in the Appendix.

6.4.8 Method Performance

Table 6.2 shows that only a few of the correct assignments are found in several methods. However, in this case, the tensor is generated from uniform random, the score error is less than 50% of the smallest assigned value (using the original assignment). From Table 6.2, the instability of this worst-case model is seen and the significant difference in assignments with a minimal difference in assignment score. Which reinforces the need for precision and recall based metrics, rather than reliance on total score. It can therefore be concluded that different optimisations have different behaviours based on the distribution of values. Approximations must be selected according to the values expected, as this can have an impact on training of larger pipelines if the assignment is non-optimal. It is also worth noting that recursive methods are generally not recommended for use with tensor accelerators as for-loops are difficult to optimise during compilation. In this case, the ability to repeatedly call it a fixed number of times on a static tensor is a significant boost and can be optimised in the same way as a typical CNN. However, they can be optimised in situations where there is a fixed or predictable number of predictions, as seen in the original work of Pair-DETR for relation prediction.

6.5 Precision in Non-Continuous Contexts

6.5.1 Why Precision Matters

The crux of linear sum assignment problems in a mathematical sense is that they cannot be solved by greedy algorithms. Many of the solutions rely on creating a sparsity over the input space. Usually done by repeatedly subtracting the minimum values, until each row and column has a 0 in it. From there, assign the locations to any row or column with exactly a single 0, and repeat until all rows and columns have an assignment. The required branching makes this process typically bound to compile C code for the CPU, requiring costly moves of data across other accelerator devices. (It should be noted that accelerator cards that postdate this work like Grace Hopper architectures seek to remove this cost). Additionally, the mathematical solution uses in-place manipulations for speed, something that might impede gradient calculation

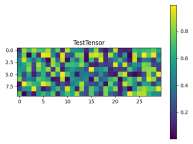
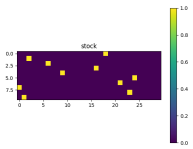
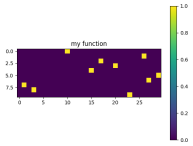
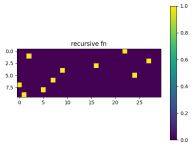
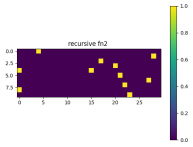
Image	Name and Description
	The random tensor for the tests to be performed, generated with Torch.rand. Random values are representative of the most chaotic assignment of a poorly trained network.
	Plain implementation of the LSA algorithm. This is the perfect assignment for the above set of values.
	Method 1: The prediction mask generated with the main GPU-based approximation.
	Method 2: The plot using a recursive function has generated the closest mask, and scores the highest of the approximations using the precision metrics. Several values can be seen to directly overlap with the ground truth.
	The prediction mask generated by an alternative algorithm that does not always result in perfect compliance with LSA. It should be noted that it has conflicts and a different set of matches with the benchmark, stock implementation of the scipy library.

Table 6.2: A Table of the different assignment matrices by method from a random input

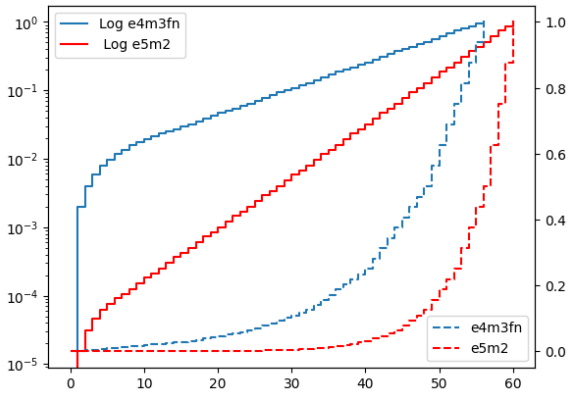


Figure 6.17: A plot showing the 128 different values expressable with each FP8 encoding scheme

for computer vision or NLP token allocation. In certain scenarios, precision may not be crucial in instances with minimal impact on the LSA score. In some situations, a precise adherence to a one-to-one mapping of a permutation matrix is unnecessary, especially during training phases if the model is poorly adjusted. In these cases, approximations such as an argmax, or a more sophisticated approach may suffice.

Until this point, LSA has been viewed within the mathematical bounds of the problem. However, with many PyTorch operations, and the non-deterministic nature of accelerations, the context must be considered.

The performance of accelerator based methods from LiSAScore across various precisions, specifically the 8-bit Floating Point (FP8) implementations of e5m2 and e4m3, and their impact on functions such as MyFunction. The attached paper demonstrates the importance of this in the context of a move to networked accelerators such as TPUs, with significant ramifications for gradient-based operations in FP8.

Perhaps most critical is the understanding that where the error is introduced is very telling in these contexts : Just because the score is nearly perfect, the difference in assignment can have some very large ramifications. In many of the tests with low precision, the model does not converge as far. This indicates that despite high accuracy and performance in approximations, there is a core function being missed consistently: assignment noise is incredibly costly.

Theoretically, where approximations improve in reduced precision, there are significant gains to be made in approximating LSA on an accelerator card. In practice, one of the issues with the manual stepped assignment is the assumption that the values are unique. In reduced precision, the algorithm fails to assign some columns due to excessive conflict caused by reduced precision and half-normal sampling. This is calculated by averaging the sum of the permutation matrix divided by the shortest

dimension across a batch.

6.5.2 Hypothesis

LSA approximations perform equally well in low-precision contexts.

In practice, the assumption of randomness of the cost matrix C is not true in computer vision settings. Although it may be for initial training steps, tracking the distribution through a DETR [16] model shows how quickly a sparse matrix appears during training. **RQ.4** is answered with the additional capabilities presented by this phenomenon and optimisations that arise: greedy algorithms in limited precision contexts are better for training, minimising memory footprint. The cost matrix generated for assignments of ground truth can be engineered for very fast performance in FP8 contexts based on greedy implementations that perform with reduced precision.

For this work, the distribution is logged of a state-of-the-art computer vision model, Pair-DETR and DETR, during training, shown in Figure 6.18.

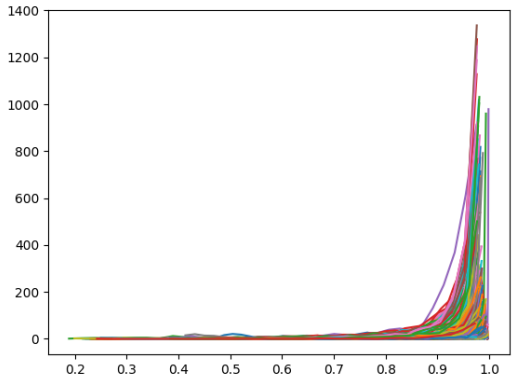


Figure 6.18: The KDE plot of values in the LSA cost matrix during Pair-DETR training

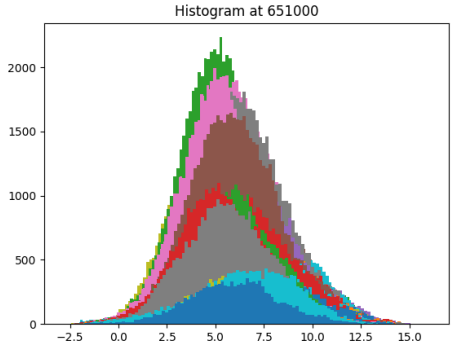


Figure 6.19: The Histogram plot of matrix values early in DETR training, showing distribution of values within the cost matrix at different steps, all tendi towards gaussian distribution

The distribution of values shows that the frequency of smaller values in the cost matrix is very heavily weighted to 1. The DETR sampling method is based on Figure 6.18, where the sampling matrix is represented as $X - |\ N(0, 0.2)|$. Runs of DETR training are monitored, noting that although the distributions develop a very slight left skew, it was not significant enough to require alterations of the supposed distribution. In later experiments, a normal distribution, a uniform distribution, and others are compared.

To apply this method in a DETR context as representative of the use case within computer vision, the normal distribution defined as $N(\mu, \sigma)$ is resampled to represent the optimal half-normal distribution. Based on the monitoring of inputs, a reasonable estimate for these values is $\mu = 5.5$ and $\sigma = 2.5$. Whilst this can come under critique for being an extreme approach, with potentially mathematically complex steps, it must be understood that this is preferable for being able to exist on an accelerator as a batch matrices operation without branching, and that resampling is relatively simple as a series of linear operations.

6.5.3 Approximations

For the purpose of this work and the goal of facilitating LSA approaches on dedicated accelerator hardware as part of a CV pipeline, the approximation function used is as defined in [86], where columns are ranked according to the cost of not selecting their maximum value. In nearly all cases, this offers a perfect approximation and trivializes the problem to selecting the column of the tensor with the maximum range in the top$-k = 2$ values, and repeating, which can be done in place using matrix operations, which naturally scale to batch operations, mitigating the overhead of repeated steps during assignment.

```
for i in range(CostMatrix.shape[small_dim]): # number of rows
        array=torch.where(mask==0,CostMatrix,ExclusionFilter)
        deltas=torch.diff(torch.topk(array,lookahead,dim=bigdim,largest=maximize).values,n=2,dim↩
            =bigdim).squeeze()
        col_index=torch.argmax(torch.abs(deltas))
```

6.5.4 Method

Precision	Float 32	Half	e4m3	e5m2
Assignments missed Half Normal(%)	0	1.3	25	32
Assignments missed Normal(%)	0	0.6	4	4

Table 6.3: Error introduced as precision decreases

The missed assignments mean that, below half precision, the models do not converge. More work is needed here to ensure that the values fall in a distribution that better fills the space. Sampling a half-normal distribution limits the runs (of a 100×100 matrix) to being very probable to contain conflicting values. There are several potential improvements, such as resampling and normalising each individual column of weights in the cost matrix; however, such approaches risk quickly increasing the computational overhead beyond the amount saved by reducing pipeline delays. Instead, using a normal distribution with mean 0 was trialled, allowing the full expressibility of reduced precision. It would be possible to attempt a wider half-normal

distribution, but a normal distribution is computationally less complicated. As the above table shows, the error is significantly reduced by the increased number of values present, and although there is still significant error, the model does meaningfully converge in reduced precision.

To maximise the efficacy of FP8, an accelerator-based implementation for LSA is used to benchmark performance. The baseline method is to find the best column to select by measuring the difference between the topk=2 value. Select the maximum value in the column, mask the corresponding rows and columns, and repeat until all assignments are made.

However, in the context of FP8, as shown in Figure 6.17, the difference between adjacent expressable values is not linear, but logarithmic. Thus, the column with the largest delta will correlate with the column with the highest value. As high values are infrequent in a sparse matrix, using argmax as a further optimisation becomes viable.

As an analogue for future experiments in the small-scale sense. Values are cast to 8 bit precision and evaluate 3 approximations that can occur on GPU, maintaining gradient, and show that in 8 bit precision they are closer to the true LSA value than in 16-bit precision. Further, the cost matrix, C, is shown to have a fixed distribution, which allows the approximations to be closer to the true value. Finally, when applied during training, the optimisations allow for faster (wall-time) convergence during training, enhanced by preserving the gradient. To offer a fair comparison, additional optimisations are included that relate to the advantage of accelerator-based hardware, such as calculating LSA concurrently across the batch.

6.5.5 Evaluation

Understanding the score can be somewhat difficult; whilst it serves well to show that methods are not equal, the complex nature of LSA means that near scores can be achieved by radically different permutation matrices.

For the purposes of understanding the comparison between approximations in limited precision, a precision and recall score is used on the permutation matrix.

$$\alpha = \frac{tr(Y^T \cdot Y_{gt})}{tr((Y_{gt})^T \cdot Y_{gt})}$$

$$\sigma = \frac{tr(Y \cdot (Y_{gt})^T)}{tr(Y_{gt} \cdot (Y_{gt})^T)}$$

where tr$(\cdot)$ returns the trace of a given matrix, and $\cdot$ a matrix multiplication. The difference between precision and recall is, therefore, the orientation of the input matrix Y. Permuting the matrix means that precision will always have an assignment in each dimension such that all dimensions must be matched. The same is not true for the recall score, which may contain non-assigned slices.

Naturally, the difference between α and σ correlates with the difference in the dimensions of the input matrix. A high relative difference will create a significant mismatch between Precision, α and Recall, σ. This is why an $F1_{LSA}$ score is introduced, similar to BERTScore, defined as:

$$F1_{LSA} = \frac{2\alpha\sigma}{\alpha + \sigma}$$

The presented formula well evaluates the comparative assignment where not all rules are adhered to, which penalises approaches that are not adherent to the problem.

In some settings, factoring in the Cost matrix C, into the evaluation can be simple as defining Y as :

$$Y = C \cdot P$$

This is rejected for existing work, due to the library incompatibility in low precision and the disparity presented across matrix size.

For experiments going forward, $F1_{LSA}$ is recommended to be scaled by matrix cost when implemented as part of wider pipelines.

6.5.6 Accelerator Approximation

The performance of each trial method is recorded and plotted with the ratio of dimensions (matrix width / height) and the $F1_{LSA}$ scores. In particular, all the graphs show that the methods perform worst when the ratio of matrix dimensions is approaching 1. This is because as width and height become equal, the fraction $\frac{min(W,H)}{WH}$ which governs the ratio of the size of the diagonal as a function of volume, is at the maximum value.

Table 6.4 shows how the performance of this approximation changes with different distributions and the imbalance of the dimensions of the matrix.

The observed imbalance serves as a clear indication that, in contexts where precision is limited, the method exhibits significant instability, yet it demonstrates substantial effectiveness in certain specific instances. When examining alterations in axis scales, the half-normal distribution emerges as the most unstable. However, considering the expressible domain of numbers previously outlined, these should be the best-performing regions. In testing, the stage of measuring the largest gap is skipped, justified by Figure 6.17. Figure 6.17. When applied correctly into the framework of DETR, the results on training time can be seen in Section 6.5.8. When applied correctly into the framework of DETR, the results on training time can be seen in Section 6.5.8.

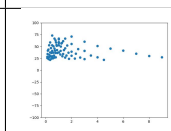
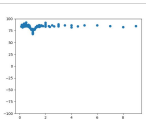
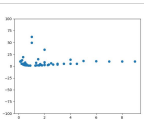
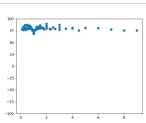
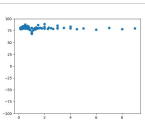
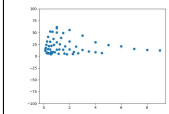
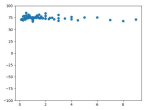
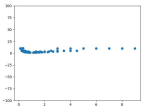
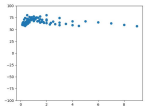
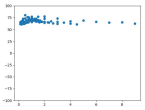
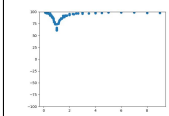
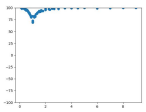
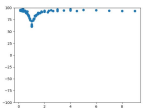
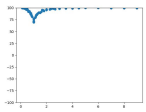
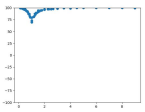
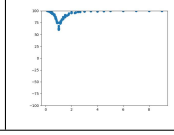
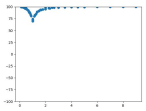
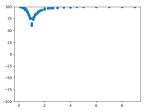
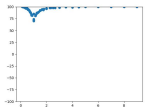
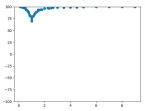
Precision	Uniform	Normal	DETR	Half-normal(0,0.3)	—X N(0,0.3)—
e4m3					
e5m2					
16					
32					

Table 6.4: $F1_{LSA}$ performance with different matrices dimensions by precision and distribution based on MyFunction, reported in [86]

6.5.7 LSA Argmax Approximation

Compared to the previous approximation, this method can be further reduced if the assertion can be made that the underlying distribution is non-negative; rather than finding the difference between the top-$k = 2$ values in each row/column of the cost matrix, a half-normal distribution means that the highest value has the largest delta. Therefore, the argmax/argmin functions find that value and repeat. Future work could even use softmax functions for gradient continuity.

Table 6.5 shows that using a simplified approximation where the top value is a good approximation of the top delta of values is a better, more consistent estimator of LSA than the primary approximation, which can score higher in some distributions but is also more sensitive to the type of random used.

The performance profile is integral to ensuring predictable behavior in low-precision computations. The approximation method offers a reduction in computational steps compared to the prior accelerator-based approach, thereby decreasing complexity in terms of both temporal and memory resources.

Table 6.5 is replicated in Table 6.6 but instead of plotting just the difference between the initial approximation and the one that selects the highest value.

From Table. 6.6, 16 and 32 bit precision show that the argmax stepped approach is a better estimator of LSA. As precision reduces, a significant drop in accuracy occurs, but is lessened by the Cost matrix being of a specific distribution aligning with the

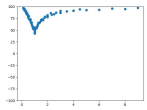
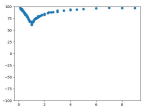
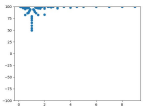
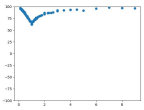
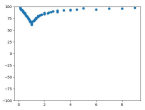
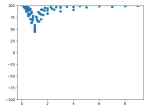
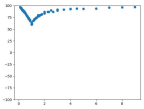
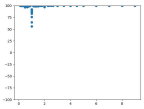
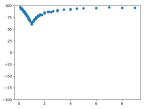
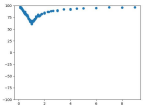
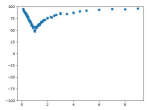
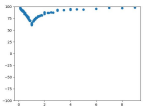
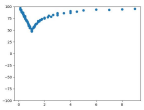
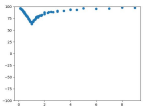
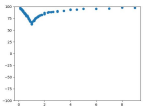
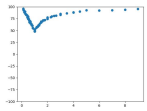
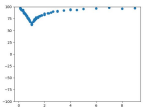
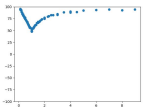
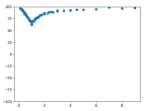
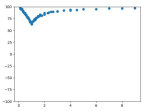
Precision	Uniform	Normal	DETR	Half-normal(0,0.3)	—X N(0,0.3)—
e4m3					
e5m2					
16					
32					

Table 6.5: $F1_{LSA}$ performance with different matrices dimensions by precision and distribution approximated taking the top value as a good approximation of the top delta of values. This is a more consistent estimator of LSA than the primary approximation, which can score higher in some distributions but is also more sensitive to the type of random used.

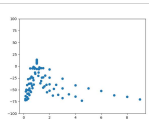
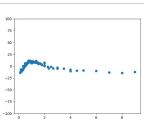
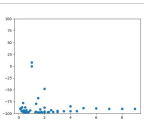
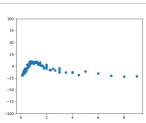
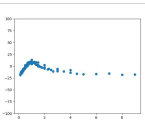
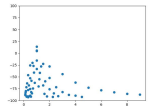
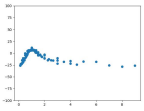
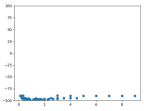
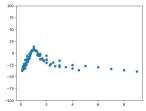
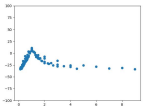
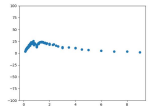
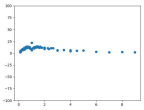
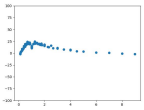
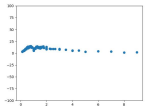
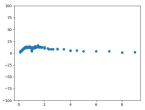
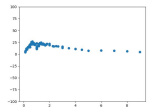
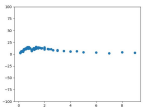
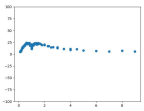
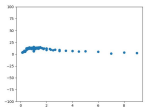
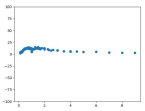
Precision	Uniform	Normal	DETR	Half-normal(0,0.3)	—X N(0,0.3)—
e4m3					
e5m2					
16					
32					

Table 6.6: The difference in $F1_{LSA}$ performance with precision and distribution using just the stepped argmax Approximation, showing that with some distributions, performance is poorer, even though reduced preciesion still outperforms others in some instances.

expressible values.

Sampling a Half-Normal distribution can even outperform the previous accelerator-based approach if the matrix has roughly comparable dimensions, where $\frac{W}{2} < H < 2W$.

6.5.8 DETR Improvement

Based on our trialled approximations, the following improvements are offered compared to the as-written implementation in DETR:

- Using the approximation of Argmin(C) as an estimator of LSA
- Resampling to a half-normal distribution and using Argmax to just select high values
- Using just argmin in our approximation of LSA, masking rows at each step
- Trialling a memory-efficient batching to process batches of images together
- The accelerator-based approximation to Linear Sum Assignment to minimize transfer latency

Method	Baseline	Argmin	Resampled and Argmax	Stepped Argmax	Extra Mem-B	Batched method
Epoch time P100	3:52 ✓	3:42 ✗	3:43 ✓	4:10 ✓	4:08 ✓	4:10 ✓
Epoch time A5000	2:50 ✓	2:29 ✗	2:31 ✓	3:10 ✓	3:06 ✓	3:01 ✓

Table 6.7: The slight optimization has a slight impact on speed on a P100 vs A5000 showing which methods converge

Metric	Baseline	Improved LSA	Approximation	Resampled with Argmax
Epoch Time	1:49	1:55	1:51	1:43
DETR reported memory use	12431	12309	11977	11962
Loss	8.8744	8.6592	8.8146	8.8498
Class Error	29.41	26.09	27.12	14.29
Loss CE	0.4526	0.4210	0.4361	0.4240
Loss BBOX	0.2917	0.2871	0.2885	0.2942
Loss GIOU	0.6587	0.6497	0.6501	0.6074

Table 6.8: Performance comparison after 90 Epochs on an RTX 3090

Table 6.7 shows how our method is slightly slower on older hardware, which uses HBM technology for improved transfer speed, which may also not be up to date for complicated statistical manipulation. The experiment was repeated on an A5000, with the same batch size to test whether this algorithmic improvement is correlated to performance. The proportional gains and losses are more significant.

There are some methods faster than the stock implementation, but our tests show that non-stepped methods do not converge as accurately, so they come with a performance cost. Comparatively, the presented approximation, with comparable accuracy, represents only 6% slowdown on conventional accelerator hardware, but a huge optimisation on TPU's, widely regarded as the way forward for hardware acceleration.

Table 6.8 shows that the training is slightly improved using the linear sum assignment, but is not statistically significant given the fluctuations per epoch. The test system ran our approach faster: possibly due to concurrent unrelated tasks making the data hand-off more costly for the stock scipy implementation, showing a significant result in reducing latency introduced by CPU-based methods. It was

noticed on some multi-gpu systems that the baseline suffered disproportionately if multiple concurrent runs were happening. The Table 6.8 has the results from a single-gpu system. Due to limited support for FP8 implementations, the values used are cast to FP8 and back to simulate reduced precision; the memory footprint and speed would be further reduced in real-world implementations.

6.5.9 Conclusion

The experiments show that in 8 bit Floating Point (FP8) precision, approximations and accelerator-based algorithms as presented in the published LiSAScore are significantly closer to the perfect solution. However, require engineering to ensure the input has an optimal distribution. The distribution of values in real-world computer vision systems transforms towards the perfect case for reduced precision implementations, making them ideal candidates. This part also demonstrated that e5m2 can result in better performance but is less practical than e4m3, partially answering the Research Question **RQ.4**, the linear sum assignment can be significantly improved.

To summarise the findings for application to computer vision systems, the optimum distribution of costs for low-precision LSA is as follows. Denoting the proposals matrix from a computer vision network as p, and the number of annotations as B, to correctly create the half-normal distribution, for a matrix of shape (p, B) the following measures are recommended. It can be safely assumed that batch annotations are typically the smaller number, which can be estimated with the product of annotations per image and batch size. Based on the density of numbers in each column, p should be less than 2^7 (128), even with a skewed sampling distribution. The values should be limited by resampling or altering the cost metrics to the specified half-normal distribution. Limiting the value to 128 ensures that the precision of FP8 is used to maximum effect. The expectation for the matrix can be considered as a product of frequency and cost; and following the prescribed formula, should be 0.24, with costs scaled suitably. To ensure saturation, the Half Normal distribution has a variance of 0.3. This gives a 0.26% chance of a value being > 0.9. For a batch size of 10 and 200 proposals, the expectation is that 5 values in the table are > 0.9, for half as many proposals, a slight increase to 0.32 provides the same number of expected values > 0.9. These guidelines provide constraints for where the accelerator-based approximations proposed will work optimally.

By removing the mathematical correctness of the function and therefore introducing an extra layer of nondeterminism, significant speedups are presented. This section has shown that approximating Linear Sum assignment can drastically speed up training, with minimal effect on model accuracy and convergence.

6.5.10 Future Improvements

The foundational principles of LSA have been elucidated, accompanied by a series of optimizations that sustain performance. The enhanced utilization of accelerator hardware and the minimization of overhead constitute a pivotal finding, demonstrating substantial improvements in training speed. Additionally, albeit to a lesser extent, this research contributes to a reduced memory footprint and less computationally intensive operations. Further progress may be achieved by refining the cost functions to generate a Cost matrix C that naturally aligns with a Normal or Half-Normal distribution, thereby diminishing the dependency on resampling, a method prevalent in the current implementation of this work.

Further work can also be done to define the resulting model error in a CV context: how $F1_{LSA}$, α and σ affect the loss and use them to scale the magnitude of the step taken.

6.5.11 Precision Limitations

This work has included a cursory study into the optimisation of a stage of computer vision frameworks. However, investigation has been limited by the scales currently available. Most machine learning libraries do not support FP8 implementations, much less as part of, and so remains largely theoretical, though with the rapid advancement of accelerator cards, this conclusion will be quick to experimentally verify going forward. Further work is required to investigate these phenomena and introduce assignment error at scale and throughout a training pipeline.

6.6 Practical Applications

This chapter has emphasized the different algorithms and nuances of the issues beneath Linear Sum assignment. The intricacies have been covered thus far in an attempt to convey that this work is evaluating the empirical method, separate from any latent or posterior task. As such, the following is a set of novel applications that could have been pursued given better research capability.

6.6.1 Loss Scaling

To answer the research question about the effect of LSA on model loss in contrastive training, it is worth considering the similar combinatorics that apply.

The properties of the permutation matrix P generated during LSA, have an analogous data structure to a linked list. The loops within this linked list which, in a random case, approximate $\frac{B}{2}$. The tendency to form loops of lengths up to B is

the same mathematics that belies the 100-prisoner problem, but can be considered an opposite case to the contrastive training uses. Contrast training points to the rare case where P is an identity matrix of size B in each dimension (and only has ones along the diagonal). From work in combinatorics, the probability of this occurring randomly in the 100-prisoner problem approaches 0 as the number of prisoners increases. A limitation of this concept is that approximations as presented would have to be restricted to those that entirely obey the rules of LSA: otherwise, the conceptual leap to treating P as a linked list and the associative properties do not conform.

6.6.2 LiSAScore

One of the significant publications to come from this work is the careful curation of LiSAScore, an improvement over BERTScore [86].

BERTScore suffers usability hurdles because of overinflated metrics. There is a significant right skew to the results compared to other token-based metrics. It should be acknowledged that token-based metrics often underreport success in well-trained models.

The work in this area identifies that the concept of precision and recall is somewhat nebulous when aiming at posterior embeddings rather than tokens, and this can often result in double-counting of frequent tokens. A similar phenomenon is caused by the special tokens added to the sequence that can easily dominate the calculation.

The use of linear sum assignment limits the counting of tokens by masking a pattern where each row and column only has a single value in it. The surprising result of this work is that all uses remove the right-skew, with the recursive set of algorithms performing closest to conventional metrics. Recursive metrics emphasize speed and optimisations over the rules of LSA, which often leads to an imbalance between Precision and Recall compared to other methods. As such, the possibility of the occasional error is balanced by some tokens, where poor matches are ignored entirely. [1]

In comparison to later applications in computer vision, the distribution of BERT latent embeddings follows a Gaussian distribution:

6.6.3 Computer Vision

Computer vision applications are perhaps the most obvious avenue for this body of work, as it is typically reliant on the Hungarian algorithm anyway.

[1] A report containing run details and full experiment write up used in the published paper can be found at https://wandb.ai/st7ma784/BertCLIPScore/reports/A-comparison-of-CLIPScore-and-Bertscore.

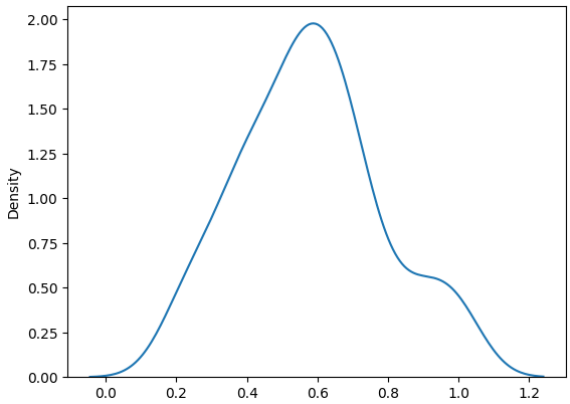


Figure 6.20: The distribution of similarities between LLM Latent embeddings

It remains to be tested how effective the recursive approximations are for training, and testing the benefits of accelerator-based approaches is beyond the remit and scales available for this body of work. However, the associated paper shows the efficacy of LSA applications with reduced precision and how the approximations demonstrated on smaller tasks can have a significant performance application in computer vision for accelerator-based approximation.

However, in lieu of demonstrable application, the bigger breakthrough of this work is not that it incrementally moves a benchmark. Instead, it is worth considering the ramifications of reimplementing a method that maintains a residual gradient.

Enabling a gradient allows each prediction to be assigned as before but also to be affected by a loss. Predictions and class logits can, therefore, be increasingly sensitive to the movement away from being candidates in the case of poor prediction, allowing inductive biases to align with the goal of NMS.

6.6.4 Bridging Computer Vision Applications

In terms of a wider pipeline in computer vision, that the Permutation matrix P is then applied back to C, means that subsequent loss functions are performed on CP. This means that the application being explored in this work, or contrastive training, can be considered as a special case of cost matrix where $C = CP$. Therefore, the subsequent discussion about calculating a permutation matrix for use in loss is interesting in both applications, because computing the Permutation matrix is effectively a null-cost - it already exists in some forms during the wider pipeline.

6.7 An Exploration of LSA for Loss

When considering the applications and notation available for the linear sum assignment, it is very quickly apparent that the assignment matrix can be considered as a permutation matrix, which is referred to in the papers generated by this work as P. In such a case, the case where each item maps to itself is unique and very special. It would be entirely reasonable to consider how this case may be learnt to maximize the values of a matrix such that the one-hot form generates this special case of P. Intrinsically, this is the same operation as contrastive loss, but initially non-continuous.

A significant advantage of the previously introduced step algorithm is the ability to carry a gradient. Gradient calculation enables a loss function, and it is no coincidence that the loss calculation in previous chapters with a diagonal staggered through an n-dimensional cube also obeys the constraints of LSA in a perfect case. This section is dedicated to exploring this approach, as it may provide a different performance characteristic and scaling rules than the prior implementation of contrastive training.

Although it improves the efficacy of training, using and maintaining calculations across n-dimensions is comparatively costly when using floating points. A way of reducing this load could be to abstract the matrix as with linear sum assignment and create a loss method based on the result.

This section emphasises using the assignment matrix itself to enhance the gradient step, rather than simply passing the allowing the direct consideration of the assignment.

6.7.1 Properties of Random LSA

The way to work out the difference between a one-hot LSA compliant array and a diagonal is to consider the array as a linked list. Treating each value as the index of the next location to check until the start location is reached. This problem is better known and documented as 'The 100 prisoner problem.'

The crux of the problem is to understand that shuffling values means that the list can be considered as a collection of loops where every location now points to another or itself (loop size = 1). Each location must be in a loop with its own value somewhere in that loop. The number of steps it takes to resolve this problem is to see the worst-case as the longest loop from the start location (to find a given number, the start location of that number must be used to ensure it is in the loop).

For loops longer than 50% of the size n, the probability that that loop is the largest is defined as $\frac{1}{k}$. As the number of prisoners in the problem becomes infinite, the plot of this tends to $y = P(\text{k is longest loop}) = \frac{1}{k}$ with range $n > k > \frac{n}{2}$. For the chance of resolving our shuffle in a number of steps,

$$P(\text{success}) = 1 - P(\text{failiure})$$

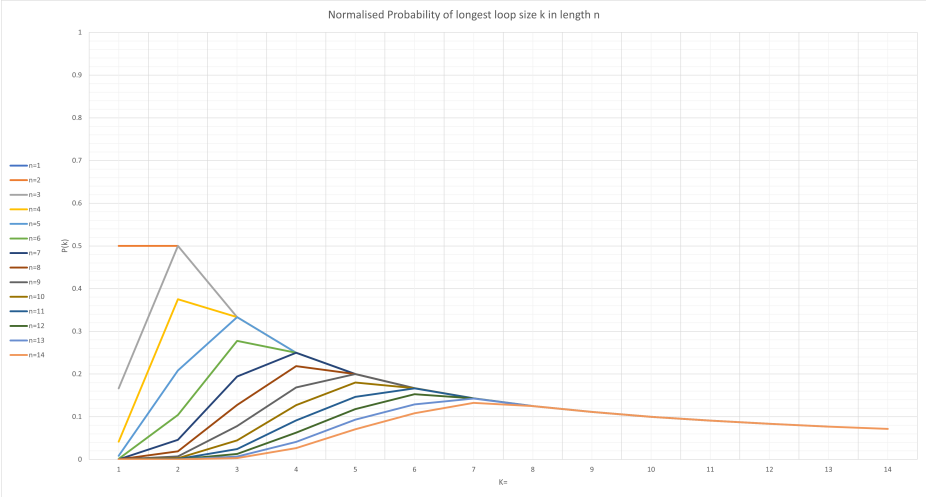


Figure 6.21: A plot of probabilities of loop size k given $0 < k < n$

where failure is defined by a loop larger than our number of guesses.

$$P(\text{success}) = 1 - \int_{\text{k}}^{n} \frac{1}{k} dx$$

We can solve this for the case where there is a half chance that all boxes resolve as

$$P(\text{failure}) = 0.5 = \int_{\text{k}}^{n} \frac{1}{k} dx$$

which is

$$ln(n) - ln(\text{k}) = 0.5$$

.

However, this still assumes that our steps, k, value is bound by $n > k > n/2$. The general plot for the range $0 > k > n$ is shown in Figure 6.21, taking continuous values. The plot in Figure 6.21 follows the general case and is plotted with continuous values. When calculated for fixed values, the plot looks like Figure 6.22b. This presents issues for gradient descent, where non-continuous values exist, but the expectation, the area between lines in Figure 6.22b, can act as a linear factor for value change.

Curiously, this tends to a constant (Golomb's constant) as size becomes infinite.

6.7.1.1 Gradient

The problem with this method of following indexes as a practical measure of resolution distance is the distinct lack of gradient.

However, as this is a very intuitive visual problem, it is worth investigating whether a neural network can approximate or intuit the distance value and, in so doing, create a continuous gradient.

Loss can be expressed in terms of the number of loops existing in a sample of n items. In the perfect case, there are n loops of length 1. In the worst case, it is one big loop of length n.

Intuitively, this results in

$$\text{loss} = \sum_{i=0}^{r} R_i^2$$

where r is the number of loops in $LSA(x)$ and R their respective lengths.

Unfortunately, this is made of discrete values and thus has no gradient per se. Therefore, a good approximation is loss $= n(E(n))$. In a random case, $textloss = E(n)^2 = n^2\lambda$, and the best case is simply $loss = n$. This can be adjusted as standard to account for batch size by dividing by n as necessary.

First, the probability that each length of loop exists:

Let $L_{k,n}$ be the number of permutations of objects n with the longest chain k. The following can be stated from literature [45]:

$$L_{1,n} = 1$$

$$L_{n,n} = (n-1)!$$

and

$$\sum_{k=1}^{k=n} L_{k,n} = 1$$

The probability of the longest chain being length k as

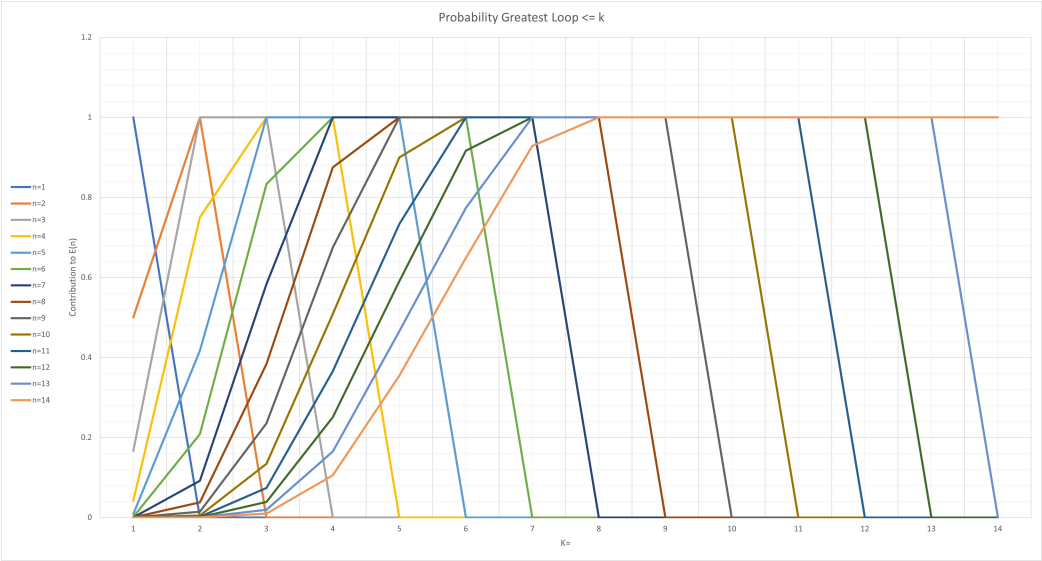
$$P(k) = \frac{L_{k,n}}{n!}$$

$L_{k,n}$ recursively can be expressed as:

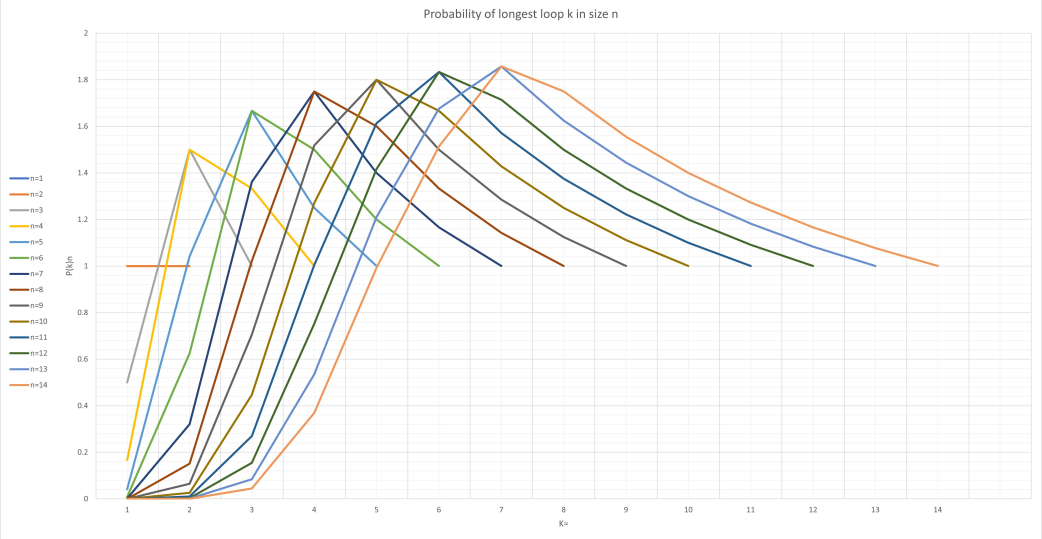
$$L_{k,n} = \sum_{j=1}^{n/k} \frac{1}{j!k^j} \cdot \frac{n!}{(n-kj)!} \sum_{t=1}^{min(k-1,n-kj)} L_{t,n-kj}$$

such that $L_{k,n}$ is a function of $L_{k',n'}$ such that $k' < k$ and $n' < n$.

Interestingly, the expectation, E_n, for a given shape as a ratio of n has a limit of the Golomb's constant, λ, as $n- > \infty$ (0.62432965....). The expectation in the infinite case could be a significant factor in defining a continuous gradient calculation. The continuous expectation is explored as a function of continuous values in the cost



(a) Plots of probability of longest loop $\leq k$ in n items



(b) A plot of the expected size of the longest loop multiplied by k

matrix and the geometry of values. Figure 6.22b demonstrates how the expectation varies with different matrix sizes, approximating the curve in Figure 6.21.

Comparing the values provided as a plot shows the tendency of the gradients to shift as theoretical probabilities become known. Statistically speaking, as individual locations are checked or assigned, the problem simplifies. It is very fast computation to check the diagonal of a permutation matrix defined previously as P. This can change the theoretical probability of the longest loop being of size n is $P(n|k)$ for n in k items to quickly take a true value of $P(n|k - tr(P))$, where $tr(\cdot)$ once again represents the diagonal of the matrix. The number of assignments that sit on the diagonal of matrix P can be quickly ignored and used to reduce the complexity of the longest loop detection.

This leads to hypotheses to test:

> The probability of certain numbers of loops of a LSA compliant matrix is predicted by the diagonal of the matrix sufficiently for loss.

To test this hypothesis, experiments will calculate loop size within the permutation matrix, and linearly scale loss accordingly. Given training works to minimize this loss, the hope is that the additive factor will allow for faster convergence in specific cases where the model is far from correctly understanding inputs.

6.7.1.2 Using LSA Loss

The base method uses the number of loops in the LSA matrix as a scaling factor for the logits.

Psuedo code is:

```
function linear_sum_assignment_steps(matrix):

    # Step 1: Apply linear sum assignment to the input matrix
    [x, y] = linear_sum_assignment(matrix)

    # Step 2: Convert assignment indices into one-hot encoded matrix
    one_hot_matrix = one_hot_encoding(y, num_classes=number_of_rows(x))

    # Step 3: Extract non-zero indices (xi, indices) from one-hot encoded matrix
    [xi, indices] = non_zero_indices(one_hot_matrix)

    # Step 4: Initialize index, counts, and foundself tensors
    index = clone(indices)
    counts = zeros_like(indices)
    foundself = zeros_like(indices)

    # Step 5: Iterate until all indices map to themselves
    while not all_true(foundself):

        # Update the index tensor by mapping through itself
        index[:] = indices[index]

        # Increment counts for elements not yet found
        for i in range(len(counts)):
            if not foundself[i]:
                counts[i] = counts[i] + 1

        # Check if the current index matches the original position
        foundself = logical_or(foundself, indices[index] == range(0, length(indices)))
```

```
# Step 6: Compute the loss based on the assigned positions and counts
loss = sum(matrix[xi, indices] * counts)

# Return the final loss
return loss
```

In this way, the loss scales by the largest loop and all values that have significant costs associated with them, reduced as they become part of smaller loops, tending to size 1. There would even be a quotient argument for subtracting $tr(x)$ from $\text{Tensor}[x_i, \text{indices}] \cdot \text{counts}$ so that the loss can become 0.

6.7.1.3 Using Expectations as Gradient

Experimentally, the ability to quickly characterize a matrix is significant in how it affects the mathematics of the problem. There are many problems with unintuitive answers yielded by subtle changes in experimental probability. In the famous Monty Hall problem, the competitor is invited to switch doors after an initial selection based on seeing another door's negative result. The revelation of another result alters the relative probability, so a competitor is counterintuitively better off switching the door.

Understanding how the probability of the assignments changes based on a simple, and cheap examination, yields the ability to predict a gradient step, based on the change in probability - the linked-list is smaller. Therefore the likelihood of a large loop is reduced. In this use case, the probability of certain permutations alters when self-referential values are removed. The remaining list has shape m such that $m = n-(tr(x))$ where $tr(x)$ is the number of points assigned on the diagonal of the original LSA matrix. This affects the probabilities in 3 ways:

- $k_{min} = 2$
- k_{max}=m
- $k \neq (m-1)$). The final remaining object cannot point to itself.
- $P(k) = \frac{L_{k,n}}{(n-1)!}$ since the number of permutations where each object cannot point to itself is fewer.

This change in probabilities creates a gradient. Furthermore, using the sum of softmaxed values assigned as probabilities rather than the count means that this gradient becomes continuous. We can also calculate the likelihood and probability changes from the integration of the graph rather than the sum of values.

6.7.1.4 Testing Experimental Probabilities

The hypothesis to test is that subtracting the sum of probabilities created by $\sum tr(x)$ from the probability distribution to produce $\frac{E_n}{n} = \lambda$ (Golomb's constant) at limit $x- > \infty$ creates a continuous gradient. Let $P(k)$ be the probability of the longest cycle length k, $dP(k)$ is calculated using the following formula:

$$dP(k) = \frac{L'_{k,m}}{(n-1)!} - \frac{L_{k,n}}{n!}$$

where $L'_{k,n}$ is the number of permutations under the new rules.

6.7.1.5 Increasing Gradient Accuracy

The values for the gradient calculation can be further calculated using the next easiest set of loops to find: $k = 2$.

Where before the can be isolated and removed with $tr(x) = \sum_{i=0}^{i=n} x_{i,i}$, the next term to remove is the loops of length 2.

$$tr'(x) = \sum x \cdot LSA(x) \cdot LSA(x)^T$$

or

$$(tr'(x) = \sum_{i=0}^{i=n} \sum_{j=-1}^{j=1} x_{i,i+j}$$

[2] This further imposes new rules on the $P(k, m)$:

- $k_{min} = 3$
- $k \neq (m-2))$. The final remaining object also cannot point to itself.
- $P(k) = \frac{L_{k,n}}{(n-2)!}$ since there are fewer permutations in which each object cannot point to itself or one another. This is only an approximation for the range $3 \leq k \leq n-2$ based on the knowledge that $P(L_{n,n}) = \frac{1}{n}$ and $P(L_{n-1,n}) = \frac{1}{n-1}$, contributing 1 to the value of E_n.

The expected loop size can therefore be differentiated as:

$$d(E_n)/dk = \int_{k=1}^{n} L_{k,n} - \int_{k=2}^{n} L'_{k,n}$$

Instead of summing the values, the integral is used to approximate the continuous values rather than the sum of discrete parts, especially where subtracting continuous probabilities may lead to fuzzy logic.

[2] Actual implementation also requires checks for array limits.

Proposition 1: Integration by Parts Formula
Integrating the function $y = L_{k,n}$ by parts:

$$\int u\,dv = uv - \int v\,du$$

This formula will be applied in the subsequent steps.

Proposition 2: Definition of u
Let u be defined as follows:

$$u = \sum_{j=1}^{\frac{n}{k}} \frac{1}{j!k^j} \cdot \frac{n!}{(n-kj)!}$$

This is the choice of u for the integration by parts in Proposition 1.

Proposition 3: Definition of dv
Let the differential dv be defined as:

$$dv = \sum_{t=1}^{\min(k-1,n-kj)} L_{t,n-kj}\,dx$$

This is the chosen form for dv, representing the portion of the function being integrated with respect to x.

Proposition 4: Application of Integration by Parts
Applying the integration by parts formula (Proposition 1), obtains the following expression for $L_{k,n}$:

$$L_{k,n} = \sum_{j=1}^{\frac{n}{k}} \left(\frac{1}{j!k^j} \cdot \frac{n!}{(n-kj)!} \right) \sum_{t=1}^{\min(k-1,n-kj)} L_{t,n-kj}$$

This expression represents $L_{k,n}$ as a sum involving $L_{t,n-kj}$.

Proposition 5: Derivative of u
Finally, the differential of u:

$$du = \frac{d}{dx} \left(\sum_{j=1}^{\frac{n}{k}} \frac{1}{j!k^j} \cdot \frac{n!}{(n-kj)!} \right) dx$$

This derivative represents the change in u with respect to x, which is required for the integration by parts.

By combining the results of these propositions, the final expression for $L_{k,n}$ as a sum of terms involving $L_{t,n-kj}$.

$$L_{k,n} = \sum_{j=1}^{\frac{n}{k}} \frac{1}{j!k^j} \cdot \frac{n!}{(n-kj)!} \sum_{t=1}^{min(k-1,n-kj)} L_{t,n-kj}$$

6.7.2 Explanation and Comparison to CELoss

As previously outlined in the evaluation of CELoss, the gradient and shape of the descent slope is crucial in any substitute function. In other words, how the embedding moves in 512-space is critical, where many small increments are possible in all directions, where there are few anchor points for first-order knowledge.

To evaluate the shape of LSA-based loss, it is worth noting the key differences between the following cases for CE loss and LSA-based loss. Crucially, this is a measure that builds on the similarity measure rather than replacing it.

Consider the following case:

The CEloss in Figure 6.23a seems to punitively account for the first column due to cross-entropy loss, and the second column is also comparatively poor.

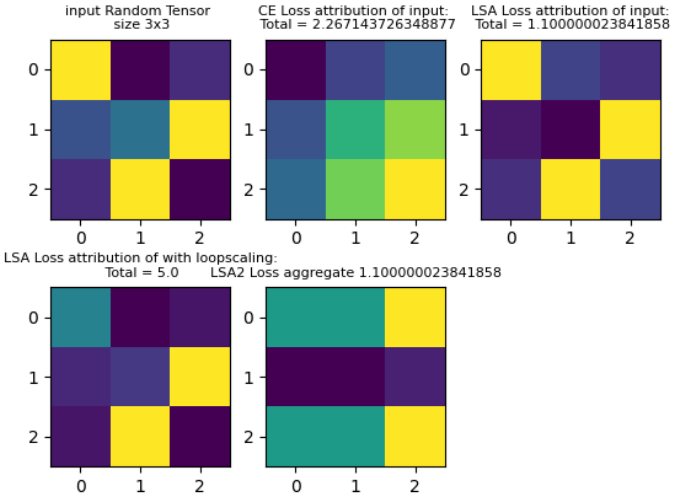
If we consider the mask that LSA generates for Figure 6.23a, it can be seen in Figure 6.23b for a worse performing loop of 3. In either case, LSA ranks Figure 6.23a as worse than Figure 6.23b. This also demonstrates the importance of including the non-selected values in LSA loss and of balancing these problems across different permutations where maximal values have to be reassigned into a new column.

6.7.3 LSA Loss Comparison During Training

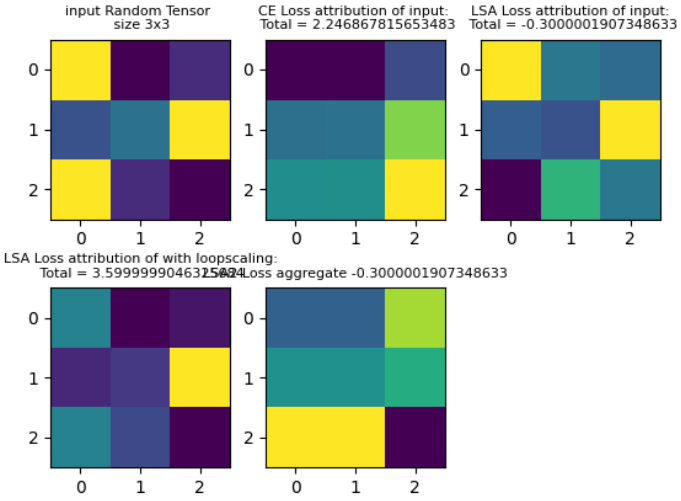
To compare the LSA-based loss, and for ease of development, the experiment is limited to the $n = 2$ case. For the sake of the test, the base contrastive code is used with the LSA method as a comparison. As shown in Figure 6.24, it is reasonable to assume that the difference in magnitude of LSA loss compared to CEloss on similarity metrics could make this an unfair comparison. Thus, an extra hyperparameter is introduced: a range of learning rates, which will yield insight on magnitude difference during comparison.

To demonstrate the comparative merits, combine several implementations of loss with the approximations and trial a small model (6 linear and activation layers). The Adam optimiser is selected for this, due to the nebulous nature of LSA and its non-stochastic nature.

To establish a benchmark for LSA-based loss,a network with Cross-Entropy loss is made as a baseline; a network comprised of 6 linear layers, showing that the loss



(a) A small scale example of the gradient produced by LSALoss when the input is complient to LSA Loss, the second row plots demonstrate the resultant vector, minimising the 2 high values not on the diagonal



(b) Compared to (a), the input has no obvious LSA solution. The second row plots demonstrate the resultant gradient towards LSA compliance.

Figure 6.23: A comparison between 2 matrix inputs showing how LSALoss responds to LSA compliant matrices (a) and not compliant matrices (b)

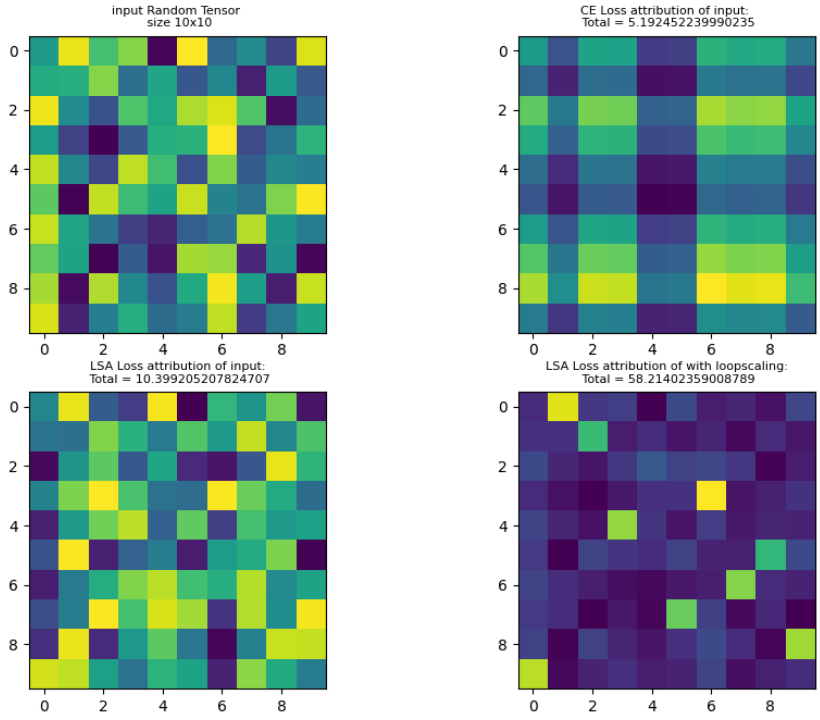


Figure 6.24: A figure showing a 10x10 plot of LSA loss, where each row may require multiple changes for LSA calculation, resulting in a more complex resultant gradient. However, the locations where the gradient is applied most each contribute significantly to LSA Error

function can correctly converge to the right output. Given a random input, the cross-entropy loss successfully converges down as shown in Figure 6.25.

This test is designed to demonstrate two things. Firstly, the outputs of a neural network do in fact converge to the correct pattern with each loss function. Secondly, this benchmark shows that there are some circumstances in which the network does not intrinsically learn to maximize the correct value based on a random input. This is reflected in the CELoss benchmark, which often plateaus at very different values. ($0.23x$) where x is the number of missing values.

The expectation is that using LSA, forcing a value in that row and column, will cause the network to converge to a state where the full diagonal is maximised.

The loss functions are summarised in Table 6.9.

These methods caused the adjustment of some approximations. The assumption

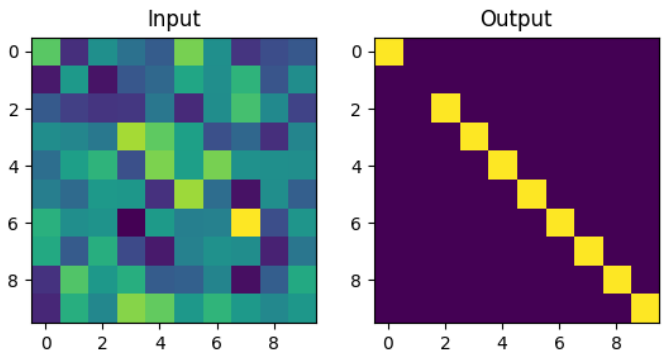


Figure 6.25: A Figure showing the result if LSALoss is applied for multiple steps on a random input, showing it converges largely towards the same resultant vector as Cross-Entropy loss, with the caveat that in some instances, some rows do not result in a one-hot vector

Method Name	Description
CE Loss	the Baseline comparison using cross-entropy loss
LSABaseLoss	Taking the sum of the assigned values, multiplied by the respective loop lengths from Permutation matrix P
LSAloss_v1	Using the comparative ratio of the assigned score compared to all other candidates. $(\sum C - \sum PC)$: $\sum PC$ assigned values. Loss is high if assigned values are not significantly higher than assigned ones.
LSAloss_v2	The Permutation matrix P is scaled according to the size of loops, then multiplied by C. Assigned values are punitively scored if they are not on the diagonal. (Noting that the adaptation of this for multiple values in each row and column means it is asymetrical, meaning that input, C gives a different result from $C.T$
LSAloss_v3	sum of —assigned values*counts— plus sum of unnassigned values. Loss is high if the values arent on the diagonal or aren't significantly higher than the others
CombinedLosses_v{}:	LSAloss_v{} + CE Loss

Table 6.9: Loss methodologies that incorporate LSA calculations

Method Name	Description
LSAFunction	GPU based masking and resampling
LSAstock	The CPU-based scipy implementation of LSA
LSAv3	The recursive LSA, based on a fixed number of reductions, and using an argmax/argmin at the end for assignment selection. This can result in empty rows and columns.
LSAv4	The same as LSAv3, with assignment selection using the top-k values where $k = \min(\text{shape})$, this is faster but assumes best convergence and can result in duplicate rows and empty rows, which causes our loss algorithms to have to be adapted to use this uncertainty.

Table 6.10: The functions in [86], tested for effectiveness in Loss

that there are internal loops requires the matrix of weights to have the same height and width. When dealing with linear sum assignment approximations, especially for uneven matrices, just dropping empty rows is insufficient in some cases to preserve this metric. Some approximations, especially recursive, do not guarantee a full assignment in either dimension. Therefore, adjustments must be made. For the counting algorithm to work, it is important that every row and column have at least 1 value. Empty rows are assigned the same values as any rows with more than 1 value. Preserving assignments and assuming minimal correction distance on errors is successful at the cost of the accuracy of the approximation functions. More work is needed to know if there are better solutions.

The LSA Functions tested are stated in Table 6.10.

6.7.4 Expectation

The test has been designed to demonstrate both the converging properties of LSA and compare performance over more than a single step to the widely accepted baseline of cross-entropy loss. Given the behaviour of LSA algorithms, and how fragile a permutation matrix can be with respect to individual values in a cost matrix, it is expected that the resultant graphs over several epochs are irregular, but ought to average out over enough random cases.

The experiment that tracks how loss boosting works over several epochs is important because it shows the cumulative direction of gradient steps towards a goal - and how that can affect loss curves. When summing cumulative steps, it is theoretically true that opposite gradients average out, but this is to neglect that this is wasted time during training, and when multiple layers of parameters are concerned,

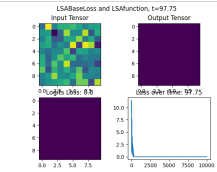
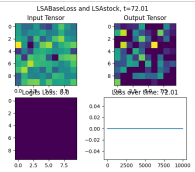
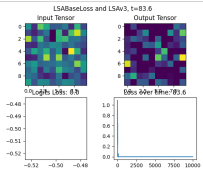
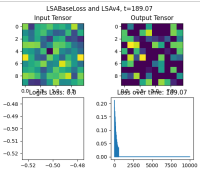
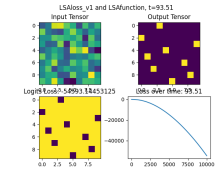
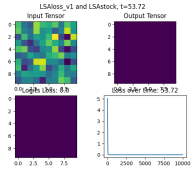
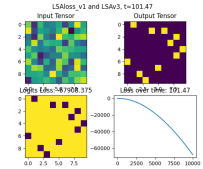
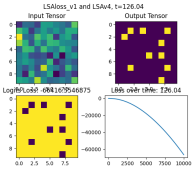
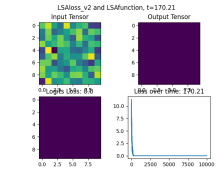
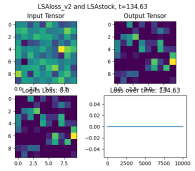
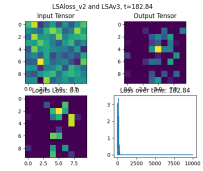
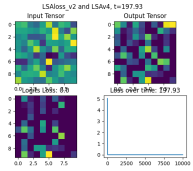
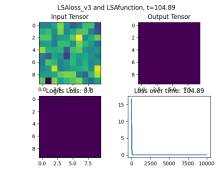
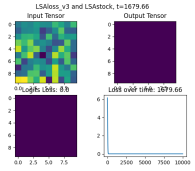
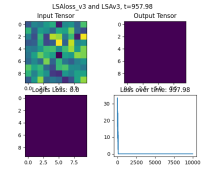
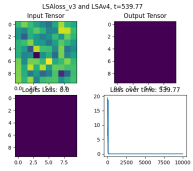
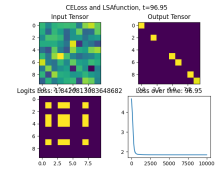
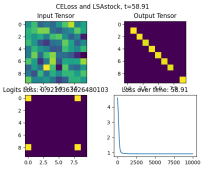
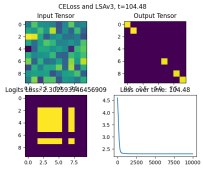
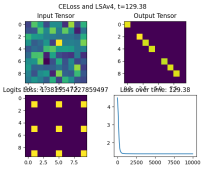
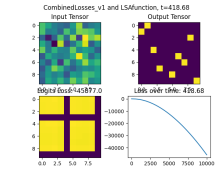
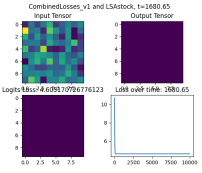
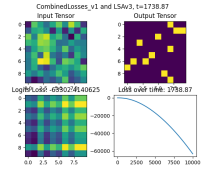
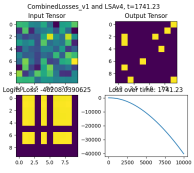
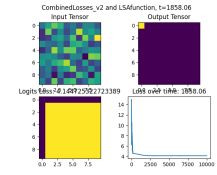
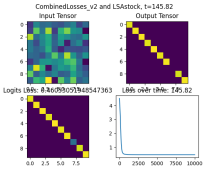
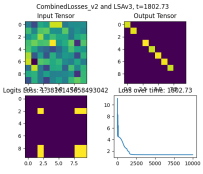
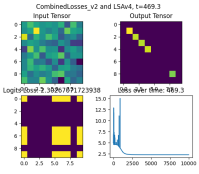
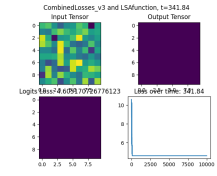
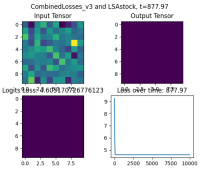
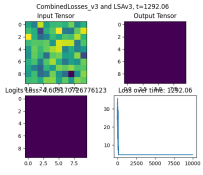
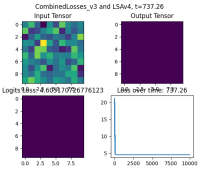
Loss	LSAFunction	LSAStock	LSAv3	LSAv4
Base LSA-Loss				
LSA loss_v1				
LSA loss_v2				
LSA loss_v3				
CE Loss				
Combined Losses_v1				
Combined Losses_v2				
Combined Losses_v3				

Table 6.11: A Table showing loss attribution, gradient and deescent curves from LSA Loss on a random input

it can be damaging for optimization schemes to have steps where no meaningful learning occurs. Therefore, the expectation of this experiment is to see that different approximation methods, although having very similar scores as previously discussed, have different performance when models converge. This is a key insight for deploying such approximations into downstream tasks like computer vision pipelines where performance in single cases does not accurately convey how model convergence is affected by the approximations' variance from the perfect solution.

6.7.5 Result Explanation

As previously illustrated in small scale in Figure 6.25 and Figure 6.24, the gradient descent of the LSA-based loss is attributable to individual parts of the input array, and this can affect the outcome over multiple steps. Thus, Table 6.11 shows the problems with using LSA in isolation for a loss function, showing how random inputs converge over multiple steps. In many cases, there is a very weak correlation in the output of the small model. Such a weak output is unsuitable for effective guiding of contrastive training given the computational expense. Whilst the loss is better for the ideal case than random, there are still mathematically superior loss cases, resulting in methods that minimize all logits rapidly descending.

The comparison between the CELoss cases, where gaps are frequent, and the combined losses, especially with $LSA\text{loss}v2$, shows a significantly better output for the stock and $LSAv3$. However, this does not pass the significance threshold, and comparing the loss graphs, they bare a more striking resemblance to the CELoss than the LSA components.

Both $LSA\text{loss}v3$ and its combined counterpart converge very quickly, resulting in very even distributions of logits, and are invariant to the different approximations of LSA. Comparatively, $LSA\text{loss}v1$ shows a promising trajectory for approximation-based approaches, but doesn't descend with the stock implementation, which demonstrates the importance of using operators capable of gradient calculation. The stock implementation also causes issues when using the base method to calculate loss. $LSAv4$ also has 2 graphs, corresponding to the base method and the $v2$-combined method, which descend at a gradual pace, thereby indicating that the gradient may be obscured during computation.

6.8 Chapter Summary

The calculation of the LSA factor, compared to CELoss, does not maintain a gradient toward the optimal value. The lack of descent slope suggests that this method may better suit reinforcement learning methods, although this would require a lot of work. A hypothesis for subsequent work would be to consider whether just minimizing

the selected values becomes significantly more effective, given the significance of the special case of loss.

This chapter has shown that there are practical limits to LSA approximations with neural networks. **RQ.4** asks whether the assignment matrix can contribute towards computational gradient. The case of the assignment matrix being able to inform the loss criterion has been explored: it is not strong enough to solely contribute to a gradient. The conclusion of this work is that it can inform the learning rate by adding a multiplier to loss; it is too computationally expensive and quickly superfluous to be meaningfully implemented in these large-scale use cases of contrastive training.

In this chapter, Linear Sum Assignment has been considered in the context of its role in the machine learning pipeline, with several approaches presented. It has been shown that reducing the precision changes how the problem can be approached, reducing branching and complexity, enabling accelerator-based approaches to become increasingly effective, reducing the overheads introduced by CPU-handoff and data transfer that can become significant overheads in multi-tenant systems.

Against the perfect case of the stock implementation, the reimplemented function with GPU acceleration performs the best. The slight deviation is due to the exceptional case where the latter assignment gradient is not considered, which is addressed in the Hungarian algorithm by assigning an adequate quantity of slices to cover any and all zeros that appear in the matrix in each step.

As GPU acceleration is considered, it can also be shown that in reduced precision, the comparative performance increase is not as big compared to the recursive family of algorithms. In both cases of FP8 (e5m2, e54m3), the performance of recursive approaches approaches the performance of the stock Hungarian algorithm and accelerated LSA functions. As all recursive approaches can be rapidly adapted to a fixed number of steps over a batched matrix, by rearranging the 'for' loops, they offer sizeable speed advantages. Especially in the context of ML contexts where matrices rapidly become saturated, acceleration with approximation is a very powerful and promising advance!

Unfortunately, present high-performance compute offerings available to researchers do not have FP8 support. So full CV benchmarks do not benefit from the same hardware support that other frameworks benefit from, meaning the improvements in speed and document any minor performance penalties for noisy assignment cannot be fairly measured.

RQ.4 Has been answered by showing that the error introduced by approximating assignments has a minimal impact due to the inconsistency of error in any given direction. Some approaches benefit massively from the increased speed of these approximations outweighing any associated cost for the related error.

Chapter 7

Conclusions

This thesis set out to address whether understanding could be tested in models that learn hidden representations, and whether there are ways of instilling learning in scalable ways to test understanding without web-scale apparatus. This thesis has shown effective algorithms for scaling down representation learning, set forth metrics for evaluating the capabilities, and then presented further advances afforded as scale increases. The remit of the work presented has broadened to include the training of the small scale models to demonstrate that they begin to demonstrate the emergent properties of a system that can be said to have a good world understanding in so much as having reliable zero-shot performance on downstream applications, and demonstrating the properties of latent space that are expected to follow.

In this thesis, along with the novel methods presented, the breakthroughs presented validate the previous theoretical works. How transformer representations embody events, concepts, and prototypical entities is core to many of these theories, and without these ideas, the presented methods would be unlikely to perform well. The efficacy of contrast training has been shown to be proportional to the number of guiding logits, even in noisy and weakly supervised settings.

7.1 Research Objectives Achieved and Questions Answered

RO.1 Present a training paradigm that is not tied to a domain prediction but can create meaningful embeddings. In curating the training paradigm that can take multiple inputs across domains, that has been trialed on toy languages, and that can operate with no assumption of posterior distribution, this work has successfully presented a novel approach that can be applied to many contexts. The topology of the latent space has been shown to encapsulate multiple orders of knowledge through

probes and zero-shot performance. The success of this method means that there will hopefully be a renewed emphasis on multiple data points to each entry, rather than the move to summaries and neural aggregations of human annotations. This work has concretely shown that all annotations are useful, which has implications for many domains like social networks and low-resource language. Though in both cases, the contemporary arrival of LLMs presents challenges that could not have been easily predicted and creates a new landscape for subsequent work.

RO.2 Show evaluation methods available where there is no clear holistic ground truth for what the model should have learnt. In profiling multiple approaches, this work has exposed several different metrics and evaluation approaches showing that CKA, probes, and down-stream applications are largely congruent in evaluation. An inter-training measure of token distribution has also been trialled and shown to be effective over a sufficient number of runs in identifying outliers in performance.

RO.3 Demonstrate the limits of scaling within the limits of 24 GPUHours, and the projected performance increase for future work. Much of this thesis has conclusions which have been governed and restricted by the incredible lack of provision. The levels at completion would have seemed comical even half a decade ago. However, the methods presented offer a tangible comparison to the methods used outside academia. This thesis has shown that through arduous re-implementation of libraries, there are many things possible within the available provision. The amount achieved with such limited resources offers a renewed challenge to organisations that require large training frameworks without considering environmental costs alongside the fiscal costs. However, it has been shown that the presented methods achieve state-of-the-art performance in training from scratch a model with strong zero-shot performance using a very small quantity of data and resources. This answers many of the concerns that inform this work around academic reproducibility, even though these methods have only received publication as a position paper: there is much prospect for future development.

In achieving these objectives, the following RQs have also been answered.

RQ.1 Demonstrate where how elements of knowledge can be isolated within a machine learning model. From the evaluation techniques and the studies into assignment, it can be clearly shown that the available representations adequately embody enough information to carry elements of semantic knowledge. The brief discussion over triples and entity-relation tuples offers a level of granularity that may never befit model representations while language is so semantically intangible.

RQ.2 Many state-of-the-art models exist beyond the training capabilities of academic replication. How can model training be reinvented to facilitate replication at a smaller scale of compute AND data? Comprehensibly and unequivocally, SOTA models could be replicated on academic scales. Unfortunately, whether (ab)using post-graduate research portfolios to explore this in lieu of correct funding and

provision remains a questionable practice. It is a sincere hope that, in spite of the success of this thesis, it does not become an exemplary case of post-graduate hyperparameter descent. This work has successfully engineered an alternative method that can be comfortably run even in the most limited of academic contexts. Despite the substantial personal financial commitments made by the author, it has been profoundly disheartening to witness how, during the conception of this groundbreaking work, several of its significant accomplishments have been eclipsed by other contemporary endeavors operating at a vastly larger scale. The frustrating reality is that these larger-scale projects, while glitzy, often lack the innovation and creative problem-solving achieved here, leveraging substantial financial resources and technological infrastructure to drive outcomes. Few of these contemporary studies were worthy of mention in related works because no changes were made to the implementations other than to throw more money and/or more data at the problem.

RQ.3 There are many advantages available to models that use weak and pairwise supervision to learn the relevant associations. Can annotation assignment have improvement on a micro scale with performance gains comparable to weak supervision on a macro scale? This work has shown that while assignment is implicit within contrastive methodologies, dynamic assignments do not produce an adequate gradient for direct training and introduce too many problems. Conversely, the way assignments are used in computer vision approaches to govern supervised annotation has been shown to have a significant impact in training. This work and the paper in the appendix have revealed that optimisations are available through approximation that can massively improve training and minimise the hurdles when moving to specialised accelerators. This work has also shown that in using assignment rules, evaluation metrics can be improved by having edge cases removed as in the paper presented at NLDB, LiSAScore.

RQ.4 Many of the semiotic studies that underpin **RO.1** point to grounding and assignment as fundamentals in attributing to a model 'understanding.' What are the impacts of approximating these assignments or using noisy approximations?

Using a noisy assignment has not been concretely tested. Whilst the theoretical approach and code base have both been rigorously developed, profiled and tested in this work, it remains astonishing that even computer vision approaches from years ago cannot be implemented at these scales.

7.2 Future Directions of Research

This work was largely inspired by the challenges faced by academic institutions today. The rate of innovation in AI, and how quickly it has captivated and accelerated thinking, has been a great leveller in some respects, and a greater divide in many others. The academic world has struggled to keep on top of reading, yet alone

the technological gulf that has opened between industry and what can be readily replicated within academic spheres.

Whilst this thesis and the work within demonstrate that cutting-edge development is possible, it is still clear that nearly half a decade on, Moore's law has not applied to equipment available to researchers, and 3 years of development are what is needed in order to begin to replicate a model that is now irrevocably entrenched in industry.

It is also worth stating that in this work, there were many contributory ideas that may once have been worthy of publication individually, and pressures academically on available resources and management have prohibited their pursuit and publication.

The ideas that can be found in the code base for this thesis are as follows.

- An improved HSIC calculation, written natively in PyTorch which allows the evaluation of a model nearly 100x faster than the available libraries, with GPU acceleration.

- An algorithm for linear sum assignment on GPU architecture. This may seem trivial, and indeed work slower, but in a datacenter or cloud-native deployment, accelerator hardware (i.e TPU) may be on a different machine to the CPU and branching, or requiring data transfer adds a prohibitive overhead.

- A novel way to train LLMs using only contrastive methods, allowing sentiment to be scored higher, a goal that has seen large amounts of traction with evaluation and loss metrics generated from BERTScore. A model that trains to new ways of expressing an idea, may be acutely more sensitive to literary style and word choice to convey nuanced or contextual ideas.

- The n-dimensional training remains unpublished beyond the theoretical paper, which received very positive reviews that asked for more evidence.

- A concrete test of Pair-DETR's methodology with extra dimensions. It is already abundantly clear that contrastive training has myriad applications spanning all domains and applications of machine learning. It would only be prudent for the algorithms selected and pioneered in this work to be put to an equally exotic set of tests across domains. During this work, code was developed to train a Pair-DETR method using this contrastive methodologies, but was unable to reach fruition due to hardware, time and RSE constraints.

- Reimplementation of pair-wise contrastive training that is significantly more efficient than other available code-bases (at time of writing).

- A template for utilising scalable HPC (High Performance Compute) offerings for machine learning workflow that has already been circulated around the N8CIR group

Many of these individually open avenues of further exploration, but primarily, the biggest single exploration direction from what has been presented is simply to scale up: showing that the assertions and beliefs within this document hold true. Chapters 3 and 4 also offer many practical applications that have not been explored, both in the domain of social media and other subjective or low-resource linguistic contexts.

7.2.1 Future Work for n-dimensional Training

A critique of the work presented in Chapter 5 is the requirement that the data have multiplicity across domains, and the benchmarks only include a dual encoder benchmark (not to be mistaken with 2 data inputs). Therefore, some future work may further explore the caveats of multiple and multimodal models. Moreover, the use of a single encoder for multiple input streams is sufficient, where the domain of each input is the same. However, comparing domains, such as a social media caption, description, and comments on social media, may cause problems. For different domains, performance may be improved with unique domain projections between feature spaces. This could be manually implemented with a single encoder, but for schemes with varying tokenizers, the implementation would require different encoders during training.

There are many different things to explore with this methodology: look at works around neuron activation such as SCRFP [65]. SCRFP pruning emphasizes the movement of neurons during training, showing that neurons that remain fixed for the entire set of inputs are less effective than moving ones in producing an effective set of weights. Works like this advocate for neurons to move, arguing that fixed neurons are most effective for single-order knowledge. As this approach cannot learn first-order knowledge, fixed neurons are of limited use and so may be pruned to improve performance or reset to promote exploration of solution spaces. During this work, many other avenues have been explored such as publications to CVPR around adversarial attack, presenting the robustness that this contrastive pretraining offers in many approaches.

7.2.2 Future Work with Linear Sum Assignment

In Chapter 6 many different approaches have been explored for both approximating the Linear Sum Assignment problem as pertains to the computer vision applications, and the applications directly to contrastive systems. As discussed in the applications section, the cost of implementing the gradient boosts is still relatively unexplored, especially where this cost matrix introduces little additional overhead.

As an evaluation metric in NLP, LiSAScore has been shown to have a better distribution than BERTScore from a usability perspective. However, BERTScore has

latterly been applied to many different contexts and places. Future work includes using LSA, and approximations, to explore whether supplementing BERTScore with LiSAScore improves the performance of advanced LLM systems, by reducing the number of tokens that can have the same activation. Whether there is a tangible difference in downstream tasks based on the uniqueness of latent representations may offer significant insights into the behaviour of large transformer-based systems.

The use of gradients in the LSA algorithm is a considerable advantage: Predicted boxes and logits can be compared against annotations using cross-entropy in a single operation. The gradient carried by LSA means that all the proposed boxes would move according to their proximity and correctness, rather than just the assigned ones. All boxes moving presents an area for further exploration of weakly supervised methods with partial or whole-image annotations, and may be a redemptive measure for 2-stage detectors when assigning non-object labels.

There is also significant room for evaluating the comparative effect of these approaches as compared to other work around computer vision. Many code bases still store the assignments as a list of indexes, rather than a permutation matrix that carries a gradient. Some work has been done in this area [92] to use permutation matrices to inform vision encoders and ensure biases are distinct in latent distributions, which has scope to be applied wider into a contrastive learning space, and latterly into other computer vision methodologies. A key future experiment would be to profile the comparison between convolutional models and transformer-based approaches with this to see if transformers are significantly more adept at learning salient features that partially encode latent permutation matrices. This would represent a significant reduction in training time and complexity for many code bases.

A conclusion of investigating LSA is that sparse tensors with permutation-based solutions manifest as very sharp parameter spaces. In practice, these challenges make gradient descent impractical without a fixed permutation matrix to descend to, as seen in other work [92]. It could therefore be proposed that this could be used as an LR scaling factor, F, for training purposes of wider frameworks. F can be defined as the sum of all loop sizes squared within P. To avoid collisions in this metric, some investigation may be needed, as previously done to discover the optimal exponent for scaling. Locating the optimal value would balance scoring single large loops higher than numerous smaller loops without scaling the LR too high. As this operation can be performed asynchronously with other operations, it is highly feasible to implement during training. Unfortunately, no time was given to explore this avenue of training on a meaningful scale, despite the potential implications for the improvement of empirical methods that rely on contrastive training.

For the purposes of future work, this problem has been abstracted into a toy machine learning repository to enable future research into learnt permutation

matrices. [1]

7.3 Final Reflections

Over this body of work, much effort has been made to define what constitutes an appropriate scale to begin investigating emergent properties of neural networks. This limit has largely been enforced by the capabilities available at an academic institution. Throughout the trajectory of this extensive body of work, it has been powerfully illustrated that small-scale research possesses the remarkable capacity to influence expansive enterprise and corporate implementations, dramatically augmenting the efficiency of systems once deployed. It remains utterly astonishing that competition and investigative endeavors are often cloaked in skepticism when undertaken on anything but a minor scale. Astonishingly, universities continue to dismiss the exploration of these phenomena on a grander scale as a legitimate research venture. Many opt for the safe harbor of believing it was unforeseeable and that the associated costs are prohibitive, instead of daring to position themselves at the vanguard of research innovation. Numerous pioneering research concepts suffer from a debilitating lack of foresight or an appreciation for the broader vision; this deficit has led to an alarming shortage of organically cultivated talent within higher education institutions, consequently stifling the influx of innovation-driven funding. It is with profound hope that the author envisions works such as this reclaiming the leadership of academic institutions in the realm of technical innovation.

[1] Repository available at https://github.com/st7ma784/EnigmaDemo

Appendix A

Visualisations

Docker image available at :

```
st7ma748/demo
```

is a web interface allowing users to play with the algorithms mentioned.

The interface offers the ability to add points to the screen, commensurate to the number of dimensions in play; this is then passed through the algorithms in this thesis. (Analogous to using them with B=1). The output scores show the similarities between points according to different outputs.

As an example, here is the output with points together, and far apart.

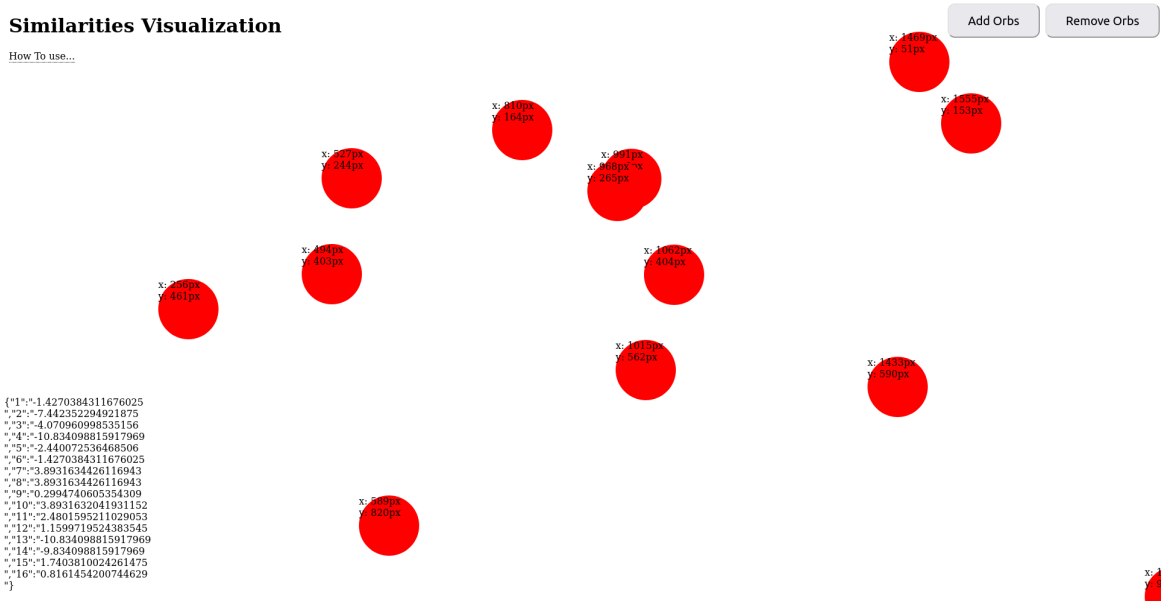


Figure A.1: Screenshot of visualisation for $n = 12$ and $F = 2$

Statistic name	returns
mean	$(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n})$
l2mean	$(\frac{\sum_{i=1}^{n} x_i^2}{n}, \frac{\sum_{i=1}^{n} y_i^2}{n})^{\frac{1}{2}}$
l3mean	$(\frac{\sum_{i=1}^{n} x_i^3}{n}, \frac{\sum_{i=1}^{n} y_i^3}{n})^{\frac{1}{3}}$
dynmean	$(\frac{\sum_{i=1}^{n} x_i^n}{n}, \frac{\sum_{i=1}^{n} y_i^n}{n})^{\frac{1}{n}}$
lsqrtmean	$(\frac{\sum_{i=1}^{n} x_i^{\frac{1}{2}}}{n}, \frac{\sum_{i=1}^{n} y_i^{\frac{1}{2}}}{n})^2$
std	$(\sqrt{\frac{\sum_{i=1}^{n}(x_i - \frac{\sum_{i=1}^{n} x_i}{n})^2}{n}}, \sqrt{\frac{\sum_{i=1}^{n}(y_i - \frac{\sum_{i=1}^{n} y_i}{n})^2}{n}})$
variance	$(\frac{\sum_{i=1}^{n}(x_i - \frac{\sum_{i=1}^{n} x_i}{n})^2}{n}, \frac{\sum_{i=1}^{n}(y_i - \frac{\sum_{i=1}^{n} y_i}{n})^2}{n}$

Table A.1: Table of useful statistics used in visualisations

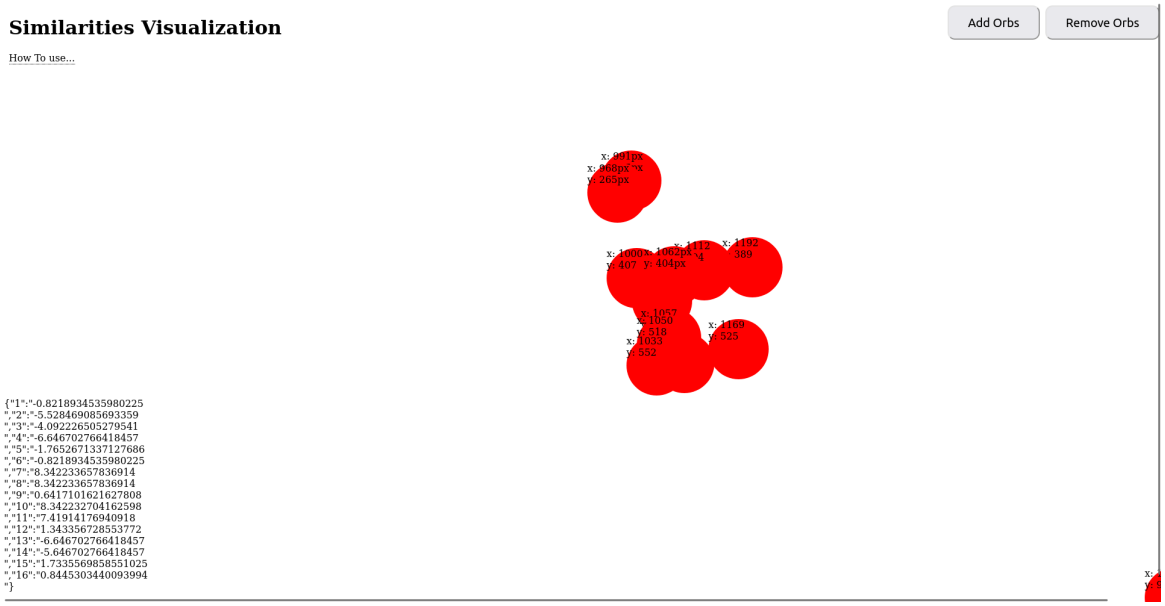


Figure A.2: Screenshot of a close clustering where $n = 12$ and $F = 2$

The Figure A.1 and Figure A.2 show screenshots from the interface of the aforementioned docker image. The red points may be freely moved around, showing how each similarity measure changes in 2D space, using the 'x' and 'y' values on the screen. This gives a very practical understanding of the outcome demonstrated in Figure 5.8 where vectors are compared by the Cartesian and cosine distances.

Appendix B

Top Performing runs for $n = 6$

The following table records the configuration and laterly the results from the top 3 runs in 6 dimensions selected based on their zero-shot topk=1 imagenet performance. In particular, performance in this domain is 10% higher than that of the in-domain COCO probe. Indicating a significant error in the class selection by taking the first annotation as indicative of the image semantics.

Other notable results include the top runs having very similar configurations in algorithms, and this resulting in similar profiles across logit regions. By comparison, the third run uses a different algorithm and has a contrasting (almost opposite) profile in these regions. Interesting for 2 reasons: firstly, the region that accounts for the average amount per region is the same! Secondly, this correlates with whether the logits are positive or negative. Method 7 therefore likely has a different profile for minimizing or maximizing, probably influenced by none of the runs using the exact labels generated.

Parameter	Run1	Run2	Run3
adam epsilon	0.00000001	0.00000001	0.00000001
batch size	10	6	8
dims	6	6	6
embed dim	512	512	512
epochs	10	10	10
exactlabels	0	0	0
gumbel	true	true	true
JSE	0	0	0
learning rate	0.0001	0.0001	0.0001
logitsversion	6	5	7
logvariance	false	false	false
maskLosses	0	0	0
meanloss	true	false	false
normlogits	false	true	true
precision	32	32	32
projection	”iinv”	”iinv”	”None”
prune	true	false	true
total steps	200,000	200,000	200,000
train batch size	10	6	8
transformer heads	16	16	16
transformer layers	5	4	6
transformer width	512	512	512
BAD logit	-85.77916717529297	-59.58275604248047	115.79511260986328
epoch	10	10	10

Metric	Run1	Run2	Run3
First Logit	-38.785	-20.045	164.94
ImProbe	0.38429	0.37463	0.37816
Logit Scale	34.094	57.450	36.336
MaskVal=6	6,218.9	3,913.3	10,566
MaskVal=20	5,800.3	3,640.4	11,742
MaskVal=34	5,376.1	3,372.7	12,917
MaskVal=48	4,945.3	3,110.7	14,093
MaskVal=84	5,040.1	3,109.4	14,093
MaskVal=98	4,608.4	2,853.2	15,268
MaskVal=162	3,832.4	2,358.7	17,619
MaskVal=258	3,996.2	2,355.4	17,619
MaskVal=272	3.5577×10^3	2.1193×10^3	1.8794×10^4
MaskVal=626	2.7367×10^3	1.4481×10^3	2.2321×10^4
MaskVal=1296	1.4472×10^3	577.97	2.7863×10^4
Mean Projection Value	-0.00011281	0.000069266	-0.000073328
Mean Validation Stock Logits	0.073854	0.12940	0.066066
Mean Loss	5.6040×10^3	3.2465×10^3	1.2770×10^4
Proportion MaskVal=6	1.1097	1.2054	0.82742
Proportion MaskVal=20	1.0350	1.1213	0.91946
Proportion MaskVal=34	0.95932	1.0389	1.0115
Proportion MaskVal=48	0.88245	0.95817	1.1036
Proportion MaskVal=84	0.89937	0.95777	1.1036
Proportion MaskVal=98	0.82234	0.87886	1.1956
Proportion MaskVal=162	0.68386	0.72656	1.3797
Proportion MaskVal=258	0.71310	0.72552	1.3797
Proportion MaskVal=272	0.63485	0.65282	1.4717
Proportion MaskVal=626	0.48834	0.44606	1.7479
Proportion MaskVal=1296	0.25825	0.17803	2.1819
TopK Imagenet	0.42088	0.41468	0.42021
TProbe	0.72574	0.70640	0.72396
Train Loss	1.9963	0.10135	0.26336
Trainer/Global Step	12,570	20,950	15,710
Val Loss-Stock	1.6494	1.4674	1.1438

Appendix C

Visualisation for LSA

Docker image available on Docker Hub as:

st7ma748/demo

The Docker image creates a web interface allowing users to play with the algorithms mentioned.

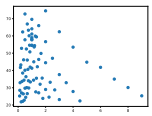
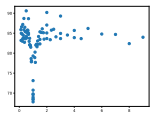
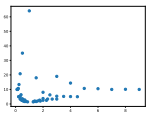
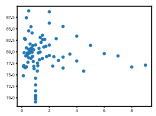
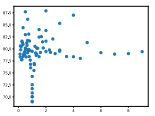
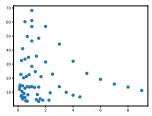
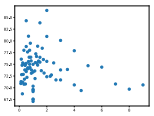
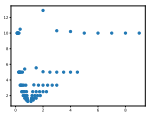
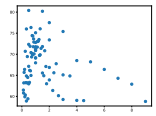
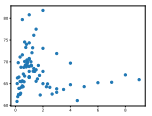
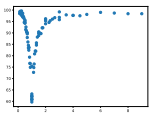
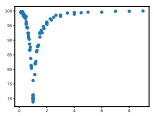
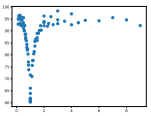
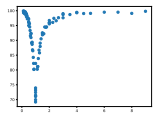
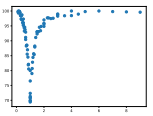
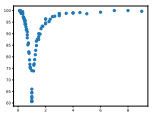
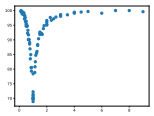
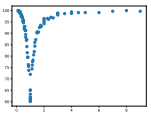
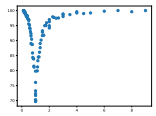
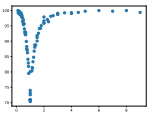
Figure C.1: Interface for trialling LSA algorithms with different precision

This visualisation provides an interface for creating a matrix ($m \times n$) and seeing the return of various LSA algorithms on it. This interface allows the different approaches to be tested on a custom matrix: showing how the assignment matrix differs with each algorithm. Further experimentation will show that using sparse values also causes the approximations to perform well against the ground truth.

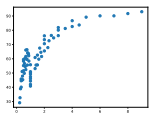
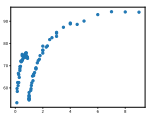
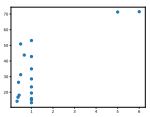
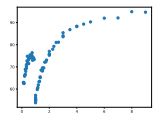
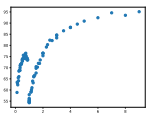
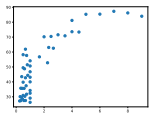
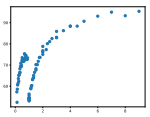
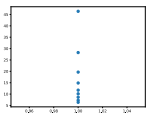
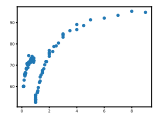
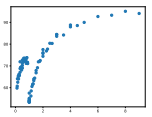
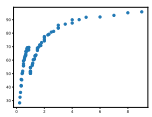
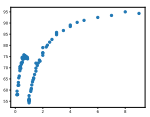
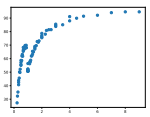
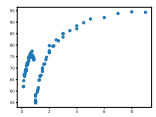
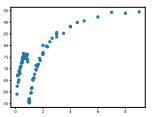
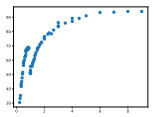
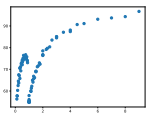
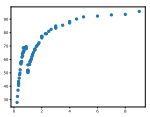
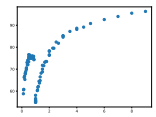
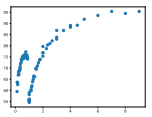
To demonstrate the performance profiles of each method and how they differ by dimensions, the following is a plot of all the methods' performance with different precisions and random distributions. Each table plots, for the given distribution and ratio between width and height, how well each method scores.

C.1 Methods

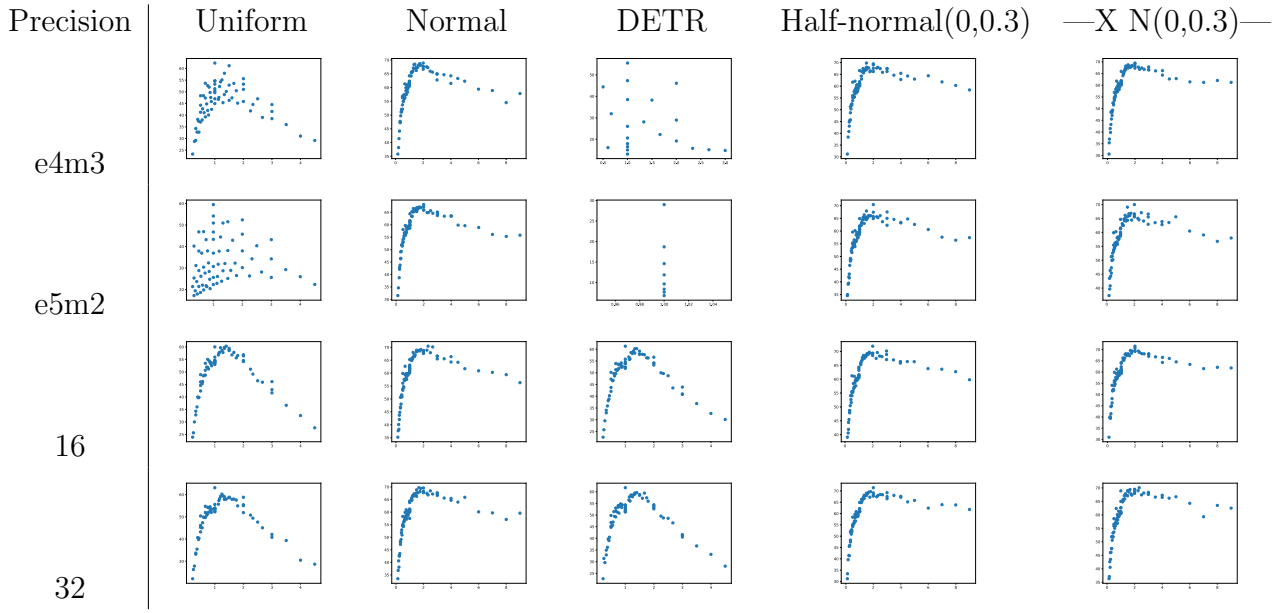
C.1.1 My Function

Precision	Uniform	Normal	DETR	Half-normal(0,0.3)	—X N(0,0.3)—
e4m3					
e5m2					
16					
32					

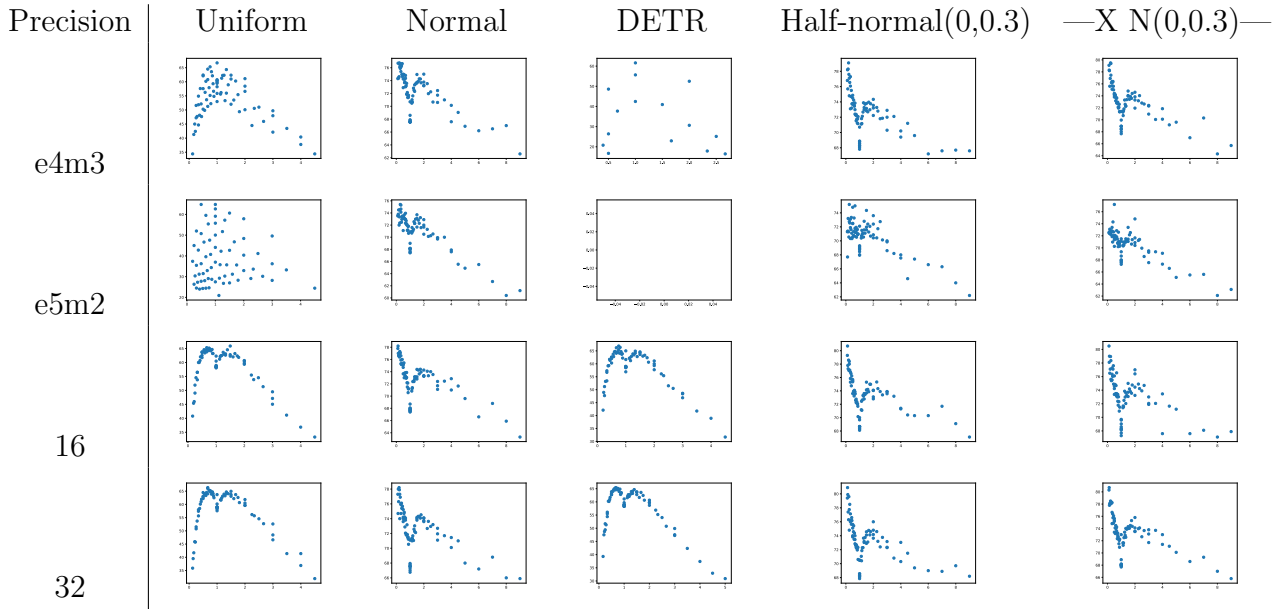
C.1.2 Recursive Fn

Precision	Uniform	Normal	DETR	Half-normal(0,0.3)	—X N(0,0.3)—
e4m3					
e5m2					
16					
32					

C.1.3 Recursive Fn v2



C.1.4 Recursive Fn v5



Bibliography

[1] Carlo Aironi, Samuele Cornell, and Stefano Squartini. "A Graph-Based Neural Approach to Linear Sum Assignment Problems". In: *International Journal of Neural Systems* 34 (Dec. 2023). DOI: 10.1142/S0129065724500114.

[2] Carlo Aironi, Samuele Cornell, and Stefano Squartini. "Tackling the Linear Sum Assignment Problem with Graph Neural Networks". In: Feb. 2023, pp. 90–101. ISBN: 978-3-031-24800-9. DOI: 10.1007/978-3-031-24801-6_7.

[3] Guillaume Alain and Yoshua Bengio. "Understanding intermediate layers using linear classifier probes". In: *arXiv preprint arXiv:1610.01644* (2016).

[4] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. "Flamingo: a visual language model for few-shot learning". In: *Advances in neural information processing systems* 35 (2022), pp. 23716–23736.

[5] Zeyuan Allen-Zhu and Yuanzhi Li. "Towards understanding ensemble, knowledge distillation and self-distillation in deep learning". In: *arXiv preprint arXiv:2012.09816* (2020).

[6] Nadja Althaus and Denis Mareschal. "Modeling cross-modal interactions in early word learning". In: *IEEE Transactions on Autonomous Mental Development* 5.4 (2013), pp. 288–297.

[7] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. "Vqa: Visual question answering". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2425–2433.

[8] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. "VQA: Visual Question Answering". In: *International Conference on Computer Vision (ICCV)*. 2015.

[9] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. "Vivit: A video vision transformer". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 6836–6846.

[10] Ratchakrit Arreerard, Stephen Mander, and Scott SL Piao. "Survey on Thai NLP language resources and tools". In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. 2022, pp. 6495–6505.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[12] Yoshua Bengio and Yann LeCun. "Scaling Learning Algorithms Towards AI". In: *Large Scale Kernel Machines*. MIT Press, 2007.

[13] Savita Bhat and Vasudeva Varma. "Large Language Models As Annotators: A Preliminary Evaluation For Annotating Low-Resource Language Content". In: *Proceedings of the 4th Workshop on Evaluation and Comparison of NLP Systems*. Ed. by Daniel Deutsch, Rotem Dror, Steffen Eger, Yang Gao, Christoph Leiter, Juri Opitz, and Andreas Rücklé. Bali, Indonesia: Association for Computational Linguistics, Nov. 2023, pp. 100–107. DOI: 10.18653/v1/2023.eval4nlp-1.8. URL: https://aclanthology.org/2023.eval4nlp-1.8/.

[14] Federico Bianchi, Giuseppe Attanasio, Raphael Pisoni, Silvia Terragni, Gabriele Sarti, and Sri Lakshmi. "Contrastive language-image pre-training for the italian language". In: *arXiv preprint arXiv:2108.08688* (2021).

[15] Christian Buck, Kenneth Heafield, and Bas Van Ooyen. "N-gram Counts and Language Models from the Common Crawl." In: *LREC*. Vol. 2. 2014, p. 4.

[16] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer. 2020, pp. 213–229.

[17] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. "Learning efficient object detection models with knowledge distillation". In: *Advances in neural information processing systems* 30 (2017).

[18] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan

Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. "Evaluating Large Language Models Trained on Code". In: *CoRR* abs/2107.03374 (2021).

[19] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. "Uniter: Universal image-text representation learning". In: *European conference on computer vision*. Springer. 2020, pp. 104–120.

[20] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. "UNITER: Learning UNiversal Image-TExt Representations". In: *CoRR* abs/1909.11740 (2019). arXiv: 1909.11740. URL: http://arxiv.org/abs/1909.11740.

[21] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. "Rethinking attention with performers". In: *arXiv preprint arXiv:2009.14794* (2020).

[22] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. "Electra: Pre-training text encoders as discriminators rather than generators". In: *arXiv preprint arXiv:2003.10555* (2020).

[23] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. "Algorithms for learning kernels based on centered alignment". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 795–828.

[24] Bo Dai, Deming Ye, and Dahua Lin. "Rethinking the Form of Latent States in Image Captioning". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[25] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. "Bridging machine learning and logical reasoning by abductive learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[26] Siddharth Dalmia, Dmytro Okhonko, Mike Lewis, Sergey Edunov, Shinji Watanabe, Florian Metze, Luke Zettlemoyer, and Abdelrahman Mohamed. "LegoNN: Building Modular Encoder-Decoder Models". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* PP (Jan. 2023), pp. 1–15. DOI: 10.1109/TASLP.2023.3296019.

[27] Siddharth Dalmia, Dmytro Okhonko, Mike Lewis, Sergey Edunov, Shinji Watanabe, Florian Metze, Luke Zettlemoyer, and Abdelrahman Mohamed. "Legonn: Building modular encoder-decoder models". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* (2023).

[28] Giannis Daras and Alexandros G. Dimakis. *Discovering the Hidden Vocabulary of DALLE-2*. 2022. arXiv: 2206.00169 [cs.LG].

[29] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. "8-bit optimizers via block-wise quantization". In: *arXiv preprint arXiv:2110.02861* (2021).

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[31] Sander Dieleman, Charlie Nash, Jesse Engel, and Karen Simonyan. "Variable-rate discrete representation learning". In: *arXiv preprint arXiv:2103.06089* (2021).

[32] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. "Attention is not all you need: Pure attention loses rank doubly exponentially with depth". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2793–2803.

[33] Lixuan Du, Rongyu Zhang, and Xiaotian Wang. "Overview of two-stage object detection algorithms". In: *Journal of Physics: Conference Series*. Vol. 1544. 1. IOP Publishing. 2020, p. 012033.

[34] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. *All nlp tasks are generation tasks: A general pretraining framework*. 2021.

[35] Kaiwen Duan, Lingxi Xie, Honggang Qi, Song Bai, Qingming Huang, and Qi Tian. "Corner Proposal Network for Anchor-Free, Two-Stage Object Detection". In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Cham: Springer International Publishing, 2020, pp. 399–416. ISBN: 978-3-030-58580-8.

[36] Dezheng Feng and Kay L O'Halloran. "Representing emotive meaning in visual images: A social semiotic approach". In: *Journal of Pragmatics* 44.14 (2012), pp. 2067–2084.

[37] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. "Devise: A deep visual-semantic embedding model". In: *Advances in neural information processing systems* 26 (2013).

[38] Kyle Gao, Yina Gao, Hongjie He, Dening Lu, Linlin Xu, and Jonathan Li. "Nerf: Neural radiance field in 3d vision, a comprehensive review". In: *arXiv preprint arXiv:2210.00379* (2022).

[39] Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. "Scaling Deep Contrastive Learning Batch Size under Memory Limited Setup". In: *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 316–321. DOI: 10.18653/v1/2021.repl4nlp-1.31. URL: https://aclanthology.org/2021.repl4nlp-1.31.

[40] Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, et al. "Llama-adapter v2: Parameter-efficient visual instruction model". In: *arXiv preprint arXiv:2304.15010* (2023).

[41] Yifei Gao, Jiaqi Wang, Zhiyu Lin, and Jitao Sang. "AIGCs confuse AI too: Investigating and explaining synthetic image-induced hallucinations in large vision-language models". In: *Proceedings of the 32nd ACM International Conference on Multimedia*. 2024, pp. 9010–9018.

[42] Carlos García. *MS-COCO-ES*. https://github.com/carlosGarciaHe/MS-COCO-ES. 2020.

[43] Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. "Dissecting Recall of Factual Associations in Auto-Regressive Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 12216–12235. DOI: 10.18653/v1/2023.emnlp-main.751. URL: https://aclanthology.org/2023.emnlp-main.751.

[44] Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. "Dissecting recall of factual associations in auto-regressive language models". In: *arXiv preprint arXiv:2304.14767* (2023).

[45] SW Golomb and P Gaal. "On the number of permutations of n objects with greatest cycle length k". In: *Probabilistic methods in discrete mathematics (Petrozavodsk, 1996)* (1997), pp. 211–218.

[46] Antônio Gomes, Ricardo Gudwin, Charbel Nino El-Hani, and João Queiroz. "Towards the emergence of meaning processes in computers from Peircean semiotics". In: *Mind & Society* 6 (2007), pp. 173–187.

[47] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. MIT Press, 2016.

[48] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. "Knowledge distillation: A survey". In: *International Journal of Computer Vision* 129.6 (2021), pp. 1789–1819.

[49] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[50] Ricardo Gudwin and João Queiroz. "Towards an introduction to computational semiotics". In: *International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2005.* IEEE. 2005, pp. 393–398.

[51] David Ha and Jürgen Schmidhuber. "World models". In: *arXiv preprint arXiv:1803.10122* (2018).

[52] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. "Mastering atari with discrete world models". In: *arXiv preprint arXiv:2010.02193* (2020).

[53] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. "A survey on vision transformer". In: *IEEE transactions on pattern analysis and machine intelligence* 45.1 (2022), pp. 87–110.

[54] Michael Hanna and Ondřej Bojar. "A fine-grained analysis of BERTScore". In: *Proceedings of the Sixth Conference on Machine Translation*. 2021, pp. 507–517.

[55] Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. "Pipetransformer: Automated elastic pipelining for distributed training of transformers". In: *arXiv preprint arXiv:2102.03161* (2021).

[56] Geoffrey Hinton. "How to represent part-whole hierarchies in a neural network". In: *Neural Computation* 35.3 (2023), pp. 413–452.

[57] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18 (2006), pp. 1527–1554.

[58] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. "Training compute-optimal large language models". In: *arXiv preprint arXiv:2203.15556* (2022).

[59] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: `2106.09685 [cs.CL]`.

[60] Seyed Mehdi Iranmanesh, Sherry X Chen, and Kuo-Chin Lien. "Pair DETR: Toward Faster Convergent DETR". In: *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2023, pp. 1–5.

[61] Seyed Mehdi Iranmanesh, Xiaotong Chen, and Kuo-Chin Lien. "Pair DETR: Contrastive Learning Speeds Up DETR Training". In: *arXiv preprint arXiv:2210.16476* (2022).

[62] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. "Perceiver: General perception with iterative attention". In: *International conference on machine learning*. PMLR. 2021, pp. 4651–4664.

[63] William James and Charles Stein. "Estimation with quadratic loss". In: *Breakthroughs in statistics: Foundations and basic theory*. Springer, 1992, pp. 443–460.

[64] Ding Jia, Yuhui Yuan, Haodi He, Xiaopei Wu, Haojun Yu, Weihong Lin, Lei Sun, Chao Zhang, and Han Hu. "DETRs With Hybrid Matching". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 19702–19712.

[65] Ziping Jiang. "On Explaining Neural Network Robustness with Activation Path". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=piIsx-G3Gux.

[66] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. "Marian: Fast Neural Machine Translation in C++". In: *Proceedings of ACL 2018, System Demonstrations*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 116–121. URL: http://www.aclweb.org/anthology/P18-4020.

[67] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. "Transformers in vision: A survey". In: *ACM computing surveys (CSUR)* 54.10s (2022), pp. 1–41.

[68] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations". In: *International journal of computer vision* 123 (2017), pp. 32–73.

[69] Tian Lan, Michalis Raptis, Leonid Sigal, and Greg Mori. "From subcategories to visual composites: A multi-level framework for object detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 369–376.

[70] Samuel Lavoie, Polina Kirichenko, Mark Ibrahim, Mahmoud Assran, Andrew Gordon Wilson, Aaron Courville, and Nicolas Ballas. *Modeling Caption Diversity in Contrastive Vision-Language Pretraining*. 2024. arXiv: 2405.00740 [cs.CV].

[71] Carlos Leandro. *Categorical semiotics: Foundations for Knowledge Integration*. 2024. arXiv: 2404.01526 [cs.AI]. URL: https://arxiv.org/abs/2404.01526.

[72] Mengyuan Lee, Yuanhao Xiong, Guanding Yu, and Geoffrey Li. "Deep Neural Networks for Linear Sum Assignment Problems". In: *IEEE Wireless Communications Letters* PP (June 2018), pp. 1–1. DOI: 10.1109/LWC.2018.2843359.

[73] Sangyun Lee. "DALLE-2". In: ().

[74] Yoav Levine, Noam Wies, Or Sharir, Hofit Bata, and Amnon Shashua. "Limits to depth efficiencies of self-attention". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 22640–22651.

[75] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *arXiv preprint arXiv:1910.13461* (2019).

[76] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models". In: *arXiv preprint arXiv:2301.12597* (2023).

[77] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 12888–12900.

[78] Xirong Li, Xiaoxu Wang, Chaoxi Xu, Weiyu Lan, Qijie Wei, Gang Yang, and Jieping Xu. "COCO-CN for Cross-Lingual Image Tagging, Captioning and Retrieval". In: *CoRR* abs/1805.08661 (2018). arXiv: 1805.08661. URL: http://arxiv.org/abs/1805.08661.

[79] Yangguang Li, Feng Liang, Lichen Zhao, Yufeng Cui, Wanli Ouyang, Jing Shao, Fengwei Yu, and Junjie Yan. "Supervision Exists Everywhere: A Data Efficient Contrastive Language-Image Pre-training Paradigm". In: *CoRR* abs/2110.05208 (2021). arXiv: 2110.05208. URL: https://arxiv.org/abs/2110.05208.

[80] Zichao Li, Cihang Xie, and Ekin Dogus Cubuk. *Scaling (Down) CLIP: A Comprehensive Analysis of Data, Architecture, and Training Strategies*. 2024. arXiv: 2404.08197 [cs.CV].

[81] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[82] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. "GPT understands, too". In: *AI Open* (2023). ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2023.08.012. URL: https://www.sciencedirect.com/science/article/pii/S2666651023000141.

[83] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. "GPT understands, too". In: *AI Open* (2023).

[84] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. "The flan collection: Designing data and methods for effective instruction tuning". In: *arXiv preprint arXiv:2301.13688* (2023).

[85] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. "Pretrained transformers as universal computation engines". In: *arXiv preprint arXiv:2103.05247* 1 (2021).

[86] Stephen Mander and Jesse Phillips. "LiSAScore: Exploring Linear Sum Assignment on BertScore". In: *Natural Language Processing and Information Systems*. Ed. by Amon Rapp, Luigi Di Caro, Farid Meziane, and Vijayan Sugumaran. Cham: Springer Nature Switzerland, 2024, pp. 249–257. ISBN: 978-3-031-70242-6.

[87] Stephen Mander and Jesse Phillips. "LiSAScore: exploring the effect of Linear Sum Assignment on BERTScore". In: *Natural Language Processing and Information Systems - 28th International Conference on Applications of Natural Language to Information Systems, NLDB 2024, Turin, Italy 2024, Proceedings*. Vol. 13914. Lecture Notes in Computer Science. Springer, 2024.

[88] Stephen Mander, Scott Piao, and Hossein Rahmani. *Contrastive Training with more data*. 2023. URL: https://openreview.net/forum?id=ZTp85mW5nFy.

[89] Stephen Mander, Scott Piao, and Hossein Rahmani. "Contrastive Training with more data". In: *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023*. Ed. by Krystal Maughan, Rosanne Liu, and Thomas F. Burns. OpenReview.net, 2023. URL: https://openreview.net/forum?id=ZTp85mW5nFy.

[90] Lev Manovich. "Defining AI arts: Three proposals". In: *AI and dialog of cultures" exhibition catalog. Saint-Petersburg: Hermitage Museum* (2019).

[91] Chengzhi Mao, Scott Geng, Junfeng Yang, Xin Wang, and Carl Vondrick. *Understanding Zero-Shot Adversarial Robustness for Large-Scale Models*. 2023. arXiv: 2212.07016 [cs.CV]. URL: https://arxiv.org/abs/2212.07016.

[92] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Roland Snoek. "Learning Permutations with gradient descent and the sinkhorn operator". In: *Proc. Int. Conf. on Learning Representations (ICLR). Vancouver, Canada. https://openreview. net/pdf*. 2018.

[93] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[94] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. *Orca: Progressive Learning from Complex Explanation Traces of GPT-4*. 2023. arXiv: 2306.02707 [cs.CL].

[95] Shikhar Murty, Pang Wei Koh, and Percy Liang. "Expbert: Representation engineering with natural language explanations". In: *arXiv preprint arXiv:2005.01932* (2020).

[96] Charlie Nash, Jacob Menick, Sander Dieleman, and Peter W Battaglia. "Generating images with sparse representations". In: *arXiv preprint arXiv:2103.03841* (2021).

[97] Minh Tuan Nguyen and Yong-Hwa Kim. "Bidirectional Long Short-Term Memory Neural Networks for Linear Sum Assignment Problems". In: *Applied Sciences* 9 (Aug. 2019), p. 3470. DOI: 10.3390/app9173470.

[98] Thao Nguyen, Maithra Raghu, and Simon Kornblith. "Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth". In: *CoRR* abs/2010.15327 (2020). arXiv: 2010.15327. URL: https://arxiv.org/abs/2010.15327.

[99] Bernardo Manzoni Palmeirim. "Stylizations of Being: Attention as an Existential Hub in Heidegger and Christian Mysticism". In: *Open Theology* 6.1 (2020), pp. 206–220.

[100] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[101] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[102] Tingting Qiao, Jing Zhang, Duanqing Xu, and Dacheng Tao. "Learn, imagine and create: Text-to-image generation from prior knowledge". In: *Advances in neural information processing systems* 32 (2019).

[103] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[104] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. DOI: 10.48550/ARXIV.2103.00020. URL: https://arxiv.org/abs/2103.00020.

[105] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. "Learning Transferable Visual Models From Natural Language Supervision". In: *CoRR* abs/2103.00020 (2021). arXiv: 2103.00020. URL: https://arxiv.org/abs/2103.00020.

[106] Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann. "Fixed encoder self-attention patterns in transformer-based machine translation". In: *arXiv preprint arXiv:2002.10260* (2020).

[107] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[108] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839 (2020), pp. 604–609.

[109] Bin Shan, Weichong Yin, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. "Ernie-vil 2.0: Multi-view contrastive learning for image-text pre-training". In: *arXiv preprint arXiv:2209.15270* (2022).

[110] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. "Neural logic reasoning". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 1365–1374.

[111] Oleksii Sidorov, Ronghang Hu, Marcus Rohrbach, and Amanpreet Singh. "Textcaps: a dataset for image captioning with reading comprehension". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer. 2020, pp. 742–758.

[112] Felix Stahlberg. "Neural machine translation: A review". In: *Journal of Artificial Intelligence Research* 69 (2020), pp. 343–418.

[113] Alane Suhr, Mike Lewis, James Yeh, and Yoav Artzi. "A corpus of natural language for visual reasoning". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2017, pp. 217–223.

[114] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. "Ernie 2.0: A continual pre-training framework for language understanding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05. 2020, pp. 8968–8975.

[115] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[116] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[117] Alexander Visheratin. "NLLB-CLIP–train performant multilingual image retrieval model on a budget". In: *arXiv preprint arXiv:2309.01859* (2023).

[118] Liwei Wang, Yin Li, Jing Huang, and Svetlana Lazebnik. "Learning two-branch neural networks for image-text matching tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.2 (2018), pp. 394–407.

[119] Fangyun Wei, Yue Gao, Zhirong Wu, Han Hu, and Stephen Lin. "Aligning Pretraining for Detection via Object-Level Contrastive Learning". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 22682–22694. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/bf5cd8b2509011b9502a72296edc14a0-Paper.pdf.

[120] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. "Transformers: State-of-the-art natural language processing". In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 2020, pp. 38–45.

[121] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR* abs/1910.03771 (2019). arXiv: 1910.03771. URL: http://arxiv.org/abs/1910.03771.

[122] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. "Robust fine-tuning of zero-shot models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 7959–7971.

[123] Chen Henry Wu and Fernando De la Torre. "A latent space of stochastic diffusion models for zero-shot image editing and guidance". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 7378–7387.

[124] Ping Xiao, Hannu Toivonen, Oskar Gross, Amilcar Cardoso, João Correia, Penousal Machado, Pedro Martins, Hugo Goncalo Oliveira, Rahul Sharma, Alexandre Miguel Pinto, et al. "Conceptual representations for computational concept creation". In: *ACM Computing Surveys (CSUR)* 52.1 (2019), pp. 1–33.

[125] Hu Xu, Saining Xie, Xiaoqing Tan, Po-Yao Huang, Russell Howes, Vasu Sharma, Shang-Wen Li, Gargi Ghosh, Luke Zettlemoyer, and Christoph Feichtenhofer. "Demystifying CLIP Data". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=5BCFlnfE1g.

[126] Alireza Zareian, Kevin Dela Rosa, Derek Hao Hu, and Shih-Fu Chang. "Open-Vocabulary Object Detection Using Captions". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 14393–14402.

[127] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. "Barlow twins: Self-supervised learning via redundancy reduction". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12310–12320.

[128] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. *Sigmoid Loss for Language Image Pre-Training*. 2023. arXiv: 2303.15343 [cs.CV].

[129] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. "Self-attention generative adversarial networks". In: *International conference on machine learning*. PMLR. 2019, pp. 7354–7363.

[130] Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. "Yin and Yang: Balancing and Answering Binary Visual Questions". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[131] Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. "Llama-adapter: Efficient fine-tuning of language models with zero-init attention". In: *arXiv preprint arXiv:2303.16199* (2023).

[132] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. "BERTScore: Evaluating Text Generation with BERT". In: *International Conference on Learning Representations*. 2020.

[133] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. "BERTScore: Evaluating Text Generation with BERT". In: *CoRR* abs/1904.09675 (2019). arXiv: 1904.09675. URL: http://arxiv.org/abs/1904.09675.

[134] Sipeng Zheng, Shizhe Chen, and Qin Jin. "Visual Relation Detection with Multi-Level Attention". In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: Association for Computing Machinery, 2019, pp. 121–129. ISBN: 9781450368896. DOI: 10.1145/3343031.3350962. URL: https://doi.org/10.1145/3343031.3350962.

[135] Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. "Can chatgpt understand too? a comparative study on chatgpt and fine-tuned bert". In: *arXiv preprint arXiv:2302.10198* (2023).

[136] Yiwu Zhong, Jianwei Yang, Pengchuan Zhang, Chunyuan Li, Noel Codella, Liunian Harold Li, Luowei Zhou, Xiyang Dai, Lu Yuan, Yin Li, and Jianfeng Gao. "RegionCLIP: Region-Based Language-Image Pretraining". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 16793–16803.

[137] Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, and Ishan Misra. "Detecting Twenty-Thousand Classes Using Image-Level Supervision". In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner. Cham: Springer Nature Switzerland, 2022, pp. 350–368. ISBN: 978-3-031-20077-9.

[138] Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, and Ishan Misra. "Detecting twenty-thousand classes using image-level supervision". In: *European Conference on Computer Vision*. Springer. 2022, pp. 350–368.

[139] Zhuofan Zong, Guanglu Song, and Yu Liu. "DETRs with Collaborative Hybrid Assignments Training". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 6748–6758.