The Effects of Pre-Training and Fine-Tuning CLIP with Domain-Specific Data

by

Jialan Wang

B.S. Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Authored by:	Jialan Wang Department of Electrical Engineering and Computer Science August 21^{th}, 2023
Certified by:	Wojciech Matusik Department of Electrical Engineering and Computer Science Thesis Supervisor
Certified by:	Ajay Daptardar Mercari US Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

The Effects of Pre-Training and Fine-Tuning CLIP with Domain-Specific Data

by

Jialan Wang

Submitted to the Department of Electrical Engineering and Computer Science
on August 11, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Mercari is an online two-sided marketplace that allows users to both sell and purchase items. To create the most efficient item listing process for the sellers and bring the most relevant items to the buyers, Mercari utilizes a pre-trained model called Contrastive Language-Image Pre-training (CLIP), famed for its exceptional zero-shot performances, to support the auto-filling feature for item listing and similar items recommendation. As this model is pre-trained on a general dataset gathered from the Internet, which likely does not have the same data distribution as Mercari's data and results in non-optimal performance, we would like to explore the possibility of pre-training or fine-tuning CLIP with Mercari's data to improve its performance within Mercari's data domain. We explore various training strategies to understand the effects of each and determine the most effective strategy. Our best-performing and most space-efficient model achieves a brand prediction top-1 accuracy of 89.34% with 49.89% coverage and a category prediction accuracy of 78.02% with 69.62% coverage, significantly outperforming the current zero-shot CLIP in brand prediction and marginally in category prediction. Moreover, it achieves this with an embedding size that is half of that of the original CLIP.

Thesis Supervisor: Wojciech Matusik
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Ajay Daptardar
Title: Machine Learning Engineering Manager, Mercari US

Acknowledgments

This thesis and the valuable co-op experience at Mercari would not be possible without the support of many people. I am truly grateful for all the assistance and knowledge I gained along the way. I would like to recognize the following people:

My co-op supervisor Ajay Daptardar, for his guidance in the entire duration of the thesis project and of this co-op experience. He offered me numerous ideas and provided me with directions for my project, and he introduced me to various new materials such as the CLIP model and libraries like ScaNN and transformers from Hugging Face. He helped me resolve issues during the experiments, and kept me on track of the project with semi-weekly syncs and timely communications.

My on-campus supervisor Prof. Wojciech Matusik, for willing to supervise my thesis. It had been unexpectedly challenging to find a thesis supervisor at MIT during the time, thus I really appreciate him for accepting me as a thesis student and providing me important advice on my project and the composition of my thesis.

Lichi Li, for being so resourceful and helpful throughout this project. He offered much knowledge to me, from inside the scope of this project to the ML and CS fields in general, and was always willing to assist me in finding utilizable resources and unblock me for the project when I ran into dead-ends. He also shed me light in the writing of the thesis, and in particular helped me with understanding the motivation and value behind this project as well as potential future directions from not only the academic standpoint but also the business standpoint.

Zainul Din and Chun An Chen, for always being responsive to my questions, giving me useful suggestions, and unblocking me from bugs. Through them, I got to learn about various embeddings compression techniques, the use of ML-related platforms and tools like Hugging Face and Vertex AI Matching Engine, and the workflow of productizing my findings and applying them in actual applications.

All my colleagues who helped me review my thesis, providing me with feedbacks and suggestions for improvement.

Kaori Iida, Kathleen M Sullivan, and Priscilla Capistrano, for enabling a smooth

arrangement and onboarding process for this 6A internship. I am thankful for their patience and support in the process despite the several times of internship time-frame changes due to various reasons.

Finally, my family, for always being supportive of me throughout my journey at MIT.

Overall, I deeply enjoyed the 6A program experience offered by MIT and Mercari, through which I gained invaluable knowledge in both academic and industrial aspects. This serves as a great transition for me from being a student to being an actual engineer by enabling me to conduct academic research while working closely with other engineers to productize the results. I would like to thank everyone who enabled this experience.

Contents

1 Introduction **15**

1.1 Motivation 16

1.1.1 Embeddings 16

1.1.2 Multimodality 17

1.1.3 Mercari Data Adaptation & Control 18

1.1.4 Zero-Shot, Pre-Training and Fine-Tuning 18

1.1.5 CLIP 19

1.2 Downstream Use Case 20

1.2.1 Retrieving Similar Items 20

1.2.2 Retrieval-Based Form Auto-Fill 22

1.2.3 Personalized Item Recommendation 23

1.2.4 Ranking 24

1.2.5 Model Interpretability Feedback 24

1.2.6 Other Integration Opportunities 24

1.3 Problem Statement 24

1.3.1 Milestones 25

1.4 Related Work 26

1.5 Thesis Organization 28

2 Setup and Pipeline **29**

2.1 Data 29

2.1.1 Text Data 31

2.1.2 Image Data 32

2.2 Training Experiment Pipeline . . . 33
2.2.1 Model Loading . . . 33
2.2.2 Modifications . . . 34
2.2.3 CLIP Training Flow . . . 36
2.2.4 Complete Pipeline . . . 37
2.2.5 Hardware Specifications . . . 38
2.3 Storage . . . 41
2.3.1 Image Data . . . 41
2.3.2 Checkpoints and Metrics . . . 41
2.3.3 Embeddings . . . 41
2.4 Evaluation . . . 43
2.4.1 Model Comparison . . . 43
2.4.2 Comparison with Current System . . . 45

3 Results and Discussion **47**
3.1 Pre-Training . . . 47
3.1.1 Results . . . 48
3.1.2 Discussion . . . 51
3.2 Fine-Tuning . . . 53
3.2.1 Results . . . 54
3.2.2 Discussion . . . 59
3.3 Comparison with the ANN Approach . . . 62
3.3.1 Results . . . 63
3.3.2 Discussion . . . 64

4 Conclusion & Future Work **65**
4.1 Conclusion . . . 65
4.2 Limitation and Future Work . . . 66
4.2.1 Larger Dataset & Coverage . . . 66
4.2.2 Further Embedding Size Compression . . . 66
4.2.3 Production Conversion & Online Test . . . 67

4.2.4 Comprehensive Hyper-Parameters Tuning 67
4.2.5 Comprehensive Image Pre-processing Techniques 67
4.2.6 Additional Text Data Inclusion 67
4.2.7 Parameter-Efficient Fine-tuning 68

List of Figures

1-1 An example of user viewing a specific item and getting recommended other similar items when scrolling down the page. 20

1-2 An illustration of ANN. 21

1-3 An example of the auto-filling feature in Mercari's item listing page. . 22

1-4 An illustration of CLIP's architecture from CLIP's paper. 27

2-1 An example entry of an item with selected fields shown. 30

2-2 Architecture of the CustomCLIP class. 36

2-3 Overview of the complete training pipeline. 37

3-1 Image-to-text top-{1, 5, 10}-accuracies of the pre-trained experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively. 49

3-2 Text-to-image top-{1, 5, 10}-accuracies of the pre-trained experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively. 50

3-3 Image-to-text mean and median rank of the pre-trained experiments by epoch in solid lines and dashed lines respectively. 51

3-4 Text-to-image mean and median rank of the pre-trained experiments by epoch in solid lines and dashed lines respectively. 52

3-5 Image-to-Text Top-{1, 5, 10}-accuracies of the fine-tuning experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively. 55

3-6 Text-to-Image Top-{1, 5, 10}-accuracies of the fine-tuning experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively. 56
3-7 Image-to-Text mean and median rank of the fine-tuned experiments by epoch in solid lines and dashed lines respectively. 57
3-8 Text-to-Image mean and median rank of the fine-tuned experiments by epoch in solid lines and dashed lines respectively. 57

List of Tables

2.1 Workbench instances configurations. 41

3.1 Configurations overview for pre-training experiments. 47
3.2 Pre-Trained Models Best Epoch Performance Comparison 50
3.3 Configurations overview for fine-tuning experiments. 58
3.4 Fine-Tuned Models Best Epoch Performance Comparison 58
3.5 Brand and category prediction performance comparison with Mercari's current model and search system using the ANN majority-vote approach, maximally duplicating the system's prediction process. 63

Chapter 1

Introduction

Mercari is a modern online two-sided marketplace platform where users can sell or purchase items. The platform helps sellers sell items that are no longer in need with some money in return and attracts buyers looking for items of all kinds, such as clothing, electronics, collectibles, handicrafts, and so on. It celebrates the "recycled economy" and promotes exchanges of pre-owned goods to discourage societal waste. Amongst its numerous features and selling points, the platform offers an accelerated listing process for sellers, a highly optimized search system with precise outputs, and a series of recommender systems for a wide array of use cases, all with the aim of delivering an efficient, simple, and fluid user experience to both, sellers and buyers.

At Mercari, there is a plethora of machine learning projects and system components powering numerous external user-facing features and internal automation. For example, one machine learning system allows end users to discover similar alternative items when viewing a specific item's detail page. Another machine learning system powers the intelligent auto-filling for the listing forms used by the seller. Yet another system powers the personalized item recommendation on the home page for buyers to browse. These systems aim to reduce user input chores, promote awareness and discovery, reduce cognitive stress over decision-making, and invite the users to enjoy the experience. These systems often involve a complex mix of microservices, productionalized machine learning model serving endpoints, retrieval indices, model training and prediction pipelines, data pre-processing and post-processing procedures,

deployment and monitoring infrastructure, inter-component communication, internal visualization and analysis, etc.

1.1 Motivation

There are many immediate or intermediate benefits that a competent machine learning model, such as those studied in this thesis, can bring to the business of Mercari. For instance:

- Improved efficiency in the search system and listing process can help buyers and sellers save time and improve the experience, which ultimately can improve user conversion rates.
- More accurate machine learning models may lead to increased exposure of relevant items to buyers who are interested.
- Models with stronger understanding of the platform users and associated items can provide better understanding and representation for potential downstream use cases, whether to directly provide better service or for internal business understanding.
- The points above can help drive better sell-through-rates and strengthen the customers' trust and confidence in Mercari's platform and its brand value.

For the reasons above, Mercari's machine learning team is interested in looking to construct a better item representation machine learning model that offers a rich understanding of the listings in their active inventory. There are several important technical considerations.

1.1.1 Embeddings

First, the new model needs to be able to provide representation embedding vectors. One of the common industry solutions to product retrieval and recommendations is to rely first on a model that can take a product's features as input, to then output a numerical vector that represents it. Subsequently, the embeddings are then inserted

into an optimized approximate-nearest-neighbor (ANN) retrieval index (e.g. [19, 11, 33, 15, 16, 49, 25, 47]) to allow for similar items retrieval using a query vector. Some examples of high level overview of this strategy applied to recommendation and ranking use cases in the industry can be found at [52, 14].

1.1.2 Multimodality

Second, the new model needs to be multi-modal in text and image. Mercari's item data consists of both text and image information, and missing out either can be insufficient or suboptimal. Text-only models rely relatively more on users to provide higher quality, non-adversarial and explicit inputs, where the structured data fields such as brand, category, color, size, and tags, can be inaccurate simply due to user input errors, while the unstructured fields such as title and description can have noisy, incoherent, incomplete or irrelevant information in practice. These can range from a series of reasons. For instance, users might have forgotten the details of a product after the original packaging or instruction manual is lost; it could be an input for the wrong item due to managing multiple similar but different listings; or the user simply fills in a guessed or random answer due to the difficulty of describing certain visual aspects of a product (e.g. a specific printed design pattern on a handbag or shirt).

The presence of the product photograph, which the model could take advantage of, offers a second source of information to not only act as a complement to existing information in the text but also as a place for cross-checking the coherency between these sources. A similar issue can happen in the reverse - an image-only model. Often, a conventional image-only model is trained in a supervised task recipe in which the text labels (e.g. category) are treated as numerical labels where the semantic information of the labels is completely lost in the translation. For instance, if we have labels "keyboard" and "mouse," they are typically converted to numbers such as "38" and "126" for an image-only model. In this case, while an image-text bi-modal model can learn to both distinguish them and capture the connection between these two items, an image-only model can only learn to recognize them as two different objects, without any knowledge of their relevance to each other. In addition, some types of

information are either too difficult or impossible to reliably infer from a photograph alone, such as the size of a shoe, storage disk size of a laptop or smartphone, etc. Using a bi-modal representation encourages a more holistic, more noise-tolerant view of the items. Furthermore, a multi-modal model offers the possibility to perform cross-modality semantic retrieval. This is not only helpful for denoising and coherency between modality information as stated above, but also opens new doors to additional use cases, where we might have partial information from the user and perform ANN search on the other modality.

1.1.3 Mercari Data Adaptation & Control

The model must show sufficient potential to adapt to Mercari's own data. Item text data at Mercari are often not syntactically correct English sentences, and are usually different from items on other marketplaces. This is in part due to what the platform permits or encourages, but also partly due to what style or intention the sellers adopt to emphasize or promote in their listings. Additionally, it is also important that the model is readily accessible to Mercari at all times (e.g. it cannot be behind an external API with limited access to its internals) to prevent unintended upstream changes that may cause unpleasant user experience downstream and permit flexible modifications in its implementation and future re-training or fine-tuning.

1.1.4 Zero-Shot, Pre-Training and Fine-Tuning

In recent literature, we observed three popular paradigms of seeking a practically competitive multi-modal model in an array of industrial applications, namely,

1. zero-shot prediction using a third-party pre-trained publicly available model [27, 28, 42, 43], or
2. prediction using an Mercari pre-trained model, or
3. prediction using a third-party pre-trained publicly available model, after it is fine-tuned using Mercari data [22, 35, 39].

Apriori it is not thoroughly clear which one among these three options would necessarily yield the best results, nor is it easy to guess how large the performance gap between them may be, as it depends on many factors, including:

- the model architecture design,
- the selection and quality of the pre-training data,
- the quality of the pre-training procedure and execution,
- the relationship and difference between the pre-training and fine-tuning data,
- the quality of the fine-tuning procedure and execution,
- the selection and design for test data, and
- the choice and design of the evaluation criteria and metrics.

Hence it is important to systematically explore these options and see which one would ultimately yield the most satisfactory results.

1.1.5 CLIP

For consistency and direct comparison, we will primarily focus on using a single model architecture design to experiment under the three training paradigms mentioned.

There are numerous multi-modal model recipes available in the literature (e.g. [1, 37, 28, 17, 53]) which can be repurposed for the current task. Among them, CLIP stands out to us for a number of reasons. Firstly, it is commercially licensed and open sourced, meaning it is accessible to us for direct usage or for further implementation or training. Secondly, it provides a simple contrastive training recipe that is not time-consuming to set up, and its simple dual encoder structure allows multiple immediately valuable downstream use cases. Most importantly, while having a relatively simple structure and not overly large in terms of parameter size compared to more recent image-text models, it is also reasonably close to the current state-of-the-art on general zero-shot prediction performance [42, 30, 29, 18]. Furthermore, there is abundant evidence that it can achieve promising fine-tuning performance (e.g. [6, 44, 32, 36, 50]). We will discuss more about CLIP in detail in section 1.3.

1.2 Downstream Use Case

There are numerous immediate use cases that can benefit from this study. We highlight a few here.

1.2.1 Retrieving Similar Items

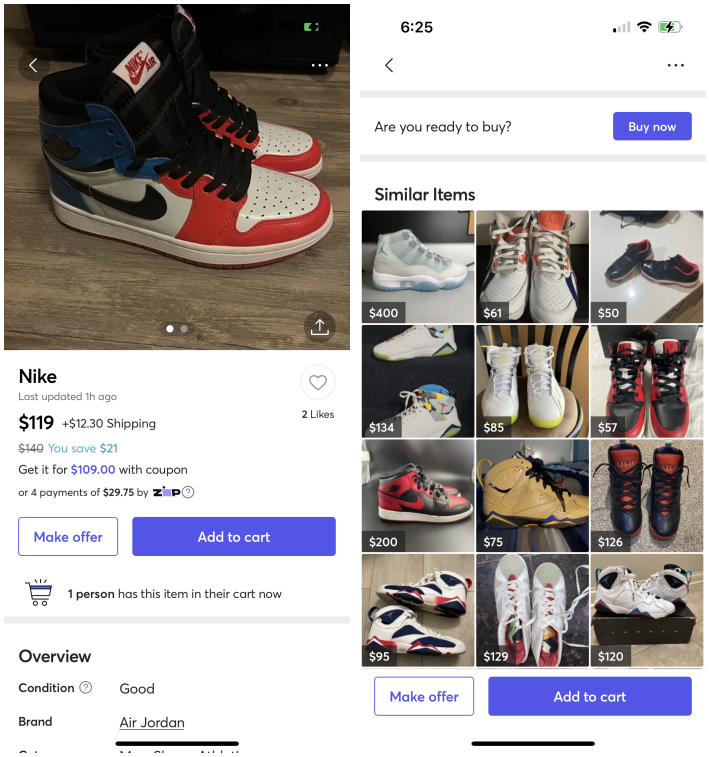

Figure 1-1: An example of user viewing a specific item and getting recommended other similar items when scrolling down the page.

When a potential buyer is viewing an item or listing x, this buyer may be interested in knowing if there are close alternatives that are more appealing, for example, lower priced, better conditions, comes in original product packaging, comes with free attachments, manufactured by a different brand, etc. In this case we can encourage inventory discovery by recommending items similar to x.

Given a machine learning model that can take features of a Mercari item x as input and an embedding vector z as an output, we can insert such embeddings into an Approximate Nearest Neighbor (ANN) vector index (e.g. [19, 11, 33, 15, 16, 49,

25, 47]) and retrieve a set of similar items x'. The overall flow can be described as follows:

1. Obtain the raw input features of the input query item x.
2. Create the embedding vector based on the model $z = f_{model}(x)$.
3. Use an ANN algorithm to find the K nearest neighbors z'.
4. With the returned items z', represented by their ids, retrieve desired information from another database in which their data are stored.

To clarify, the ANN algorithm looks for the K nearest neighbors $\{z'\}_{1:K}$ which have the shortest distances away from the query item embedding vector z. Figure 1-2 provides an illustration of the process.

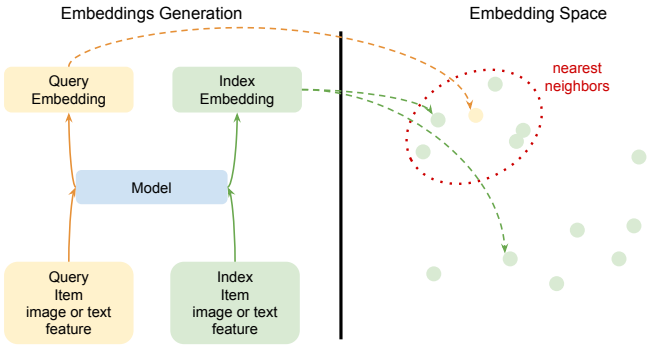

Figure 1-2: An illustration of ANN.

With this formulation, we can retrieve K items similar to x. In practice, if we already observe that a buyer is interested in x, then this mechanism can help the buyer discover more alternatives, some possibly more interesting than x while being context-relevant. This has many potential benefits, for instance:

1. increase the chance that the buyer can see a more “optimal” item that leads to a purchase,
2. accelerate the speed (and possibly volume) at which the items are sold due to the increased exposure,
3. increase the satisfaction rate of the buyers as the system promotes more desirable and "optimal" items to them, and

4. increase the chance for buyers to discover expanding interests when the retrieval is more variable (yet still contextually relatable).

1.2.2 Retrieval-Based Form Auto-Fill

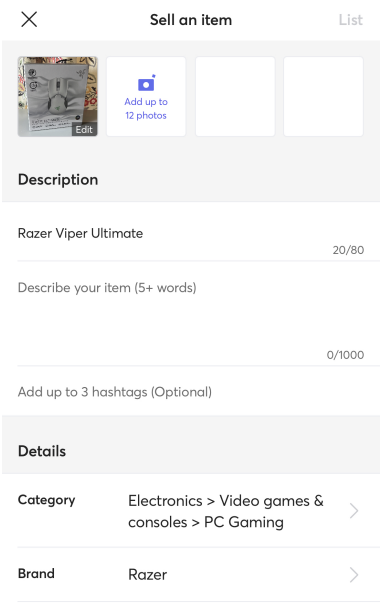

Figure 1-3: An example of the auto-filling feature in Mercari's item listing page.

When a seller creates a listing on the Mercari platform, a form appears for the seller to fill in order to provide details about their item. The form generally asks for photographs and a mix of structured and unstructured text data fields. To provide convenience to the seller via automation, we can produce information auto-filling after they provide partial information.

Let us define the set S of F information fields any single Mercari item can maximally have, $\{x_1, x_2, \cdots, x_F\}$, including multiple photographs or text fields as some of them. As the user input partial information by filling some of the F fields, we can split the set S into two sets: A set $S_{available} = \{x_1, x_2, \cdots\}$ of available inputs already provided by the user, and another set $S_{remaining} = \{\cdots, x_{F-1}, x_F\}$ currently remaining empty until the user explicitly fills them later. In this case we can build a system that takes $S_{available} = \{x_1, x_2, \cdots\}$ as input and generate predictions for fields within $S_{remaining} = \{\cdots, x_{F-1}, x_F\}$ as output, so that we can generate predicted answers

that the users may potentially intend to input.

Following a similar recipe as the similar item use case above, we can construct such a system that generates predictions based on its closest neighbors by using the embedding and ANN steps in the same way as above, and then generate the $S_{remaining} = \{\cdots, x_{F-1}, x_F\}$ responses by deriving from the neighboring items close in distance away from x.

A system like this can bring many business benefits, such as:

1. reduce the explicit effort and time in inputting information from the seller,
2. reduce the cognitive stress and learning curve for the seller (e.g. from reading and choosing the right category),
3. reduce the chance of seller giving up on filling the item details,
4. improve the chance of correctness, completeness and higher quality in the item information when facing some of the user input noise,
5. improve sellers' operating efficiency as small or individual businesses, and
6. improve Mercari's brand image and trust.

1.2.3 Personalized Item Recommendation

In many scenarios, Mercari offers personalized item recommendations for buyers to discover, explore and consider buying. While such machine learning systems rely on the knowledge of user's past and recent activities, it is generally expected to be beneficial if this system can be built on high-quality item representations, by which the system can better learn the nuance of users' interests.

By using an accurate, powerful foundational item representation model for embedding generation, we can create composite machine learning models where the high-quality item embeddings are inputs to other models to then learn in conjunction with user activities. This may increase the precision and recall of the personalized recommender systems.

1.2.4 Ranking

Given a list of Mercari items to show the potential buyer, we can also determine the expected optimal order or ranking to display in a way such that the user is more likely to engage with some of the items. In this case, a high-quality item embedding model can provide better signals for a ranking model to learn from.

1.2.5 Model Interpretability Feedback

Internally, Mercari frequently conducts studies and investigations to constantly improve its platform. One way a high-quality embedding model can be useful is to utilize interpretability methods (e.g. saliency [26], SHAP [31]) to help understand what portions within text or image of the items contribute useful information of defining a listing and leading to its long term outcome. These can be internally used as feedback signals for Mercari developers to improve systems, platform user interface, user guidance, and more.

1.2.6 Other Integration Opportunities

There are many more user interfaces of different functionality on the Mercari platform where recommendations of all kinds may seek usage of a solid item embedding model from this study, such as shopping cart recommendations, post-sales events, anti-fraud detection, and so on. Certain use cases may demand minor tweaks or modifications to the outputs of the model (e.g. applying filters on the resulting items), or join with other specialized machine learning models for other uses. We will omit the details here.

1.3 Problem Statement

This research aims to investigate the effects of pre-training and fine-tuning CLIP with Mercari's data, and explore the possibility of improving CLIP's performance within Mercari's data domain through pre-training and fine-tuning. Specifically, we would

like to conduct a series of CLIP pre-training and fine-tuning experiments with various settings in hyper-parameters and configurations to attempt to maximally adapt CLIP to Mercari's context. From these experiments, we seek to both understand the effects brought by these settings and determine the best strategy as well as the best performed pre-trained or fine-tuned model that can potentially be employed by Mercari for various applications.

To realize these experiments, we first gather and pre-process an appropriate training and test set for the experiments and other necessary datasets for model evaluation. Then, we construct the complete training pipeline through the use of the publicly available *open_ clip*[1] library and necessary implementations. Finally, we set up virtual machines and environments to run the experiments. Upon completion of the experiments, we compute appropriate evaluation metrics for each of the models for model comparison and results analysis. Additionally, we evaluate the best models from the experiment against the CLIP model currently employed by Mercari in actual applications to determine if they can truly bring improved performances to Mercari's current prediction system.

1.3.1 Milestones

The main responsibilities of this project are as follows:

- Extract and pre-process a training set and a test set for training and evaluation use during the experiments and two other sets for top-50 brands prediction evaluation and top-100 categories prediction evaluation respectively.
- Adapt the available training pipeline from the *open_ clip* library by implementing new classes and methods or modifying existing codes.
- Determine the appropriate hardware configurations and environment setup for the experiment, and set up the virtual machines and virtual environments accordingly to enable the experiments to be performed.
- Determine the set of hyper-parameters and training configurations to explore,

[1] https://github.com/mlfoundations/open_clip

and conduct and monitor the experiments.

- Compute the necessary evaluation metrics to compare the resulting models among themselves and with the zero-shot CLIP.
- Evaluate the best models from the experiments against Mercari's current prediction system, which involves constructing searchers for the models, embeddings retrieval, and implementation of the ANN majority-vote approach for prediction.

1.4 Related Work

The research concerns two main aspects, with the first being the practice of model fine-tuning. Fine-tuning of a pre-trained model is the continuation of training of a previously trained model with a new set of data that might or might not have the same distribution as that the model was pre-trained on, and there might be modifications or additions to the model's architecture depending on the targeted task of the resulting model. It has long been practiced for model adaptation to a different data distribution or task, and it is a prevalent technique in the machine learning field, especially with the rise of NLP in which many language models are pre-trained first with general language data and then fine-tuned to specific downstream tasks, like ELMo[41] and GPT-2[43]. Model fine-tuning can be as simple as loading the pre-trained weights to the model before training it with the new data set when the targeted task does not require much modifications or additions to the model, such as this CLIP fine-tuning experiment for satellite images and captions matching by Arto et. al.[2]. Some fine-tuning strategies employ a more deliberate approach of freezing certain layers of the model to prevent unwanted knowledge loss while learning new knowledge in targeted areas [23]. Many other complex and advanced approaches seek to either maximally preserve the model's robustness after fine-tuning [51] or optimize the efficiency aspect of fine-tuning [13, 5].

The other aspect is the history and studies of CLIP[42] usage and performance optimization. The creation of CLIP was inspired by the work of Li et. al. [24], which

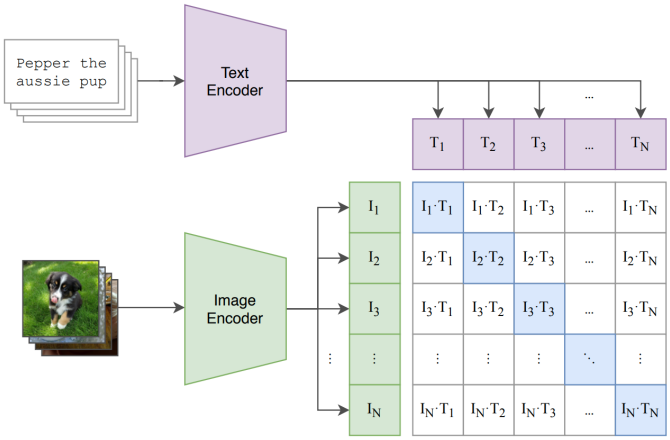

Figure 1-4: An illustration of CLIP's architecture from CLIP's paper.

showed the outstanding zero-shot transfer ability of a model when it was learning using natural language supervision. Based on this work, CLIP followed a similar approach by using captions as labels and learning to match the images to the captions. The architecture of CLIP is shown in fig. 1-4. It takes in a set of image and caption pairs, encodes them with the image and text encoder respectively, and then attempts to correctly match the pairs by computing the pair-wise cosine similarity scores of their embeddings. Trained on a 400M image-caption dataset collected from the Internet and using natural language supervision, CLIP demonstrated excellent zero-shot performance and was the state-of-the-art for zero-shot image classification and image-caption matching at the time. With the ability to directly compare images and texts, it allows for a wide range of applications like image search by text or image and image classification tasks. Given its well-known superb zero-shot performance, much of the subsequent research on CLIP focused on implementing additional procedures or modules to work directly with pre-trained CLIP, without additional training on it [45, 7, 12, 8, 9, 38, 8]. Studies regarding fine-tuning of CLIP came later, adapting CLIP to data within a specific domain [2, 40] or further enhancing CLIP's performance on a specific task [44, 10, 36, 50, 32]. For instance, Dong et. al. [6] found performance improvements by fine-tuning the CLIP's vision encoder with the ImageNet dataset for image classification, proving the potential behind fine-tuning CLIP despite prior research at the time showed uncleared or mixed results regarding the

performance of fine-tuned CLIP. There are also fine-tuning optimization approaches for large models like CLIP, such as WiSE-FT [51], which aims to fine-tune large models like CLIP while preserving the overall robustness through ensembling the weights of the zero-shot and the fine-tuned model.

1.5 Thesis Organization

Firstly, the thesis provides details regarding the preparations and setup for the experiments. After this, for both pre-training and fine-tuning experiments, we present the configurations of the experiments and then the evaluation and comparison of the resulting models. We will discuss the potential implications of the results as well as possible future work as extensions to this research.

Chapter 2 presents a complete picture of the setup and the procedures of the fine-tuning and training experiments. These include data extraction and pre-processing, experiment environment, hardware usage, libraries and necessary modifications to build a complete pipeline, checkpoints and embeddings storage, and evaluation methods.

Chapter 3 describes in detail the different experiments conducted for pre-training and fine-tuning CLIP, and delivers the complete performance results from the experiments. The effects of different investigated hyper-parameters or configurations on the model training will be discussed, along with uncertainties that are potentially worth investigating in future work.

Chapter 4 concludes the major findings and the best performance achieved from the experiments. In addition, we will elaborate on the limitations in the current research and propose potential future study.

Chapter 2

Setup and Pipeline

There are four major steps in preparing for and conducting CLIP training and fine-tuning experiments: data extraction and preparation, training experiment pipeline, model and embeddings storage, and models evaluation and comparison. Details for each step are provided in the remainder of this chapter.

2.1 Data

As the goal of the research is to obtain better representations for items to improve search and prediction accuracy, user data are not of our concern and we will be extracting data concerning only the listed items. Mercari stores a number of pieces of text information for each item ever listed on the platform as well as its available images uploaded by the seller, all also retrievable through crawling an item's webpage. An example record of an item is shown in Fig. 2-1, with selected relevant fields displayed. Some of the fields can be empty for some items, either because they are not applicable to the items or the information was not filled in by their sellers. The image shown is the original picture of the item uploaded by the seller, retrieved through the "image_url" field.

For training and fine-tuning the model, we utilized a set of approximately 7.3 million items created between 2021/09/30 and 2022/09/29, whose images had been readily proportionally resized to a smaller dimension of 299 by a larger dimension

Field	Value
name	Adidas Track Jacket in Yellow
brand name	Adidas
L0 category	Men
L1 category	Athletic apparel
L2 category	Jackets
category id	000
brand id	00
color	yellow
size	XL (46-48)
condition	Good
image url	https://mercari-images.global.ssl.fastly.net/photos/m37610026924_1.jpg

Figure 2-1: An example entry of an item with selected fields shown.

for other previous training tasks. They were first chosen due to the fact that their images are smaller than the original images, and thus more efficient to work with. Secondly, this set of items consisted of items from the top (most common) 3000 item brand-category combinations, with approximately even distribution across these combinations. Using the data of the most frequent items to train and fine-tune the model can potentially improve the model's performance on the largest set of items across the platform. This set is then split into a train test and a test set in a 99:1 ratio by taking 1% of items from each brand-category combination to build the test set. At the end of the data cleaning process, there are a total of 7,282,074 items and 72,810 items in the training and test set respectively.

Aside from the test set described above, two other datasets with different item distributions are also used for comparison of the performances of the trained and fine-tuned models. One of them is a set of items from the top-50 brands and the other from the top-100 categories. They are to gauge the prediction ability of the models on item brands and categories respectively. These datasets are exclusive to the test set to avoid bias.

Finally, to compare the best performed fine-tuned or pre-trained model from experiments with the model and the matching engine currently used by Mercari, we

retrieve a set of approximately 60 million items, created between 2022/10/01 and 2023/05/01, to be put into the index of a searcher we build using my best performed model and a publicly available library called ScaNN [11], which will be elaborated in Section 2.4. Then, another set of 10,000 items created between 2023/05/10 and 2023/05/11 is used for evaluation of brand and category prediction using the searcher. These two sets are mutually exclusive to avoid bias caused by predicting the brand and category of an item which is already in the index of the search engine.

2.1.1 Text Data

There are many pieces of text data for an item, and which ones to be used or included as part of the captions for the images can likely affect the performance of the resulting model, provided that the CLIP's text tower actually learns semantic information from the captions. While the first thought could be that including more information in the captions can help the model learn richer and more detailed features, these text data can be noisy as they are raw user inputs, thus concatenating many fields could potentially introduce more noise to the captions. Therefore, we would like to consider constructing the captions using just one or some of the following 5 fields:

- `name`
- `brand_name`
- `L0_category`
- `L1_category`
- `L2_category`

where `name` is the title of an item, `brand_name` is the name of the item's brand, and the three levels of category names are the categories of the item with increasing specificity. They are chosen because they contain the most basic but indicative information of an item and they are non-empty for all of the items. Furthermore, brand and category predictions are two of the main tasks we want the model to perform, so it is important for the model to learn from data including these pieces of information.

To keep the data format consistent, all text fields will be turned into lower case

and special or multi-space characters will be replaced with a single space. Commas and semicolons are also replaced by a single space. When constructing a caption from several fields, they will be concatenated with a single space separating each field.

2.1.2 Image Data

We first filtered the train and test datasets for corrupted images. Given part of the experiments involves exploring which image transformations yield the best performance, images are first resized to 224x224 in two ways:

- proportional resizing and then center-cropping
- plain resizing, disregarding original proportion

The reason behind cropping the images before training instead of on the fly is that loading larger images on the GPU takes up more space, resulting in a smaller maximum batch size allowed. After this pre-processing, to understand how additional transformations affect the results, some experiments will perform an extra random crop and resizing during each epoch.

Images for the train and test sets were initially stored on a cloud database called Filestore[1] and they were retrieved during training by mounting the database to the Google Vertex AI notebook where the experiments were run. However, this turned out to be a serious bottleneck for batch loading due to the network limit, resulting in significant training inefficiency. We managed to drastically speed up this process for the experiments by transferring all the images first to Google Cloud Storage and then to a local directory in the notebook using multiprocessing. The time consumption improved from more than 5 days per epoch to approximately 2 hours per epoch with our hardware specifications.

[1] https://cloud.google.com/filestore

2.2 Training Experiment Pipeline

A complete and flexible implementation for the training experiment pipeline enables training CLIP with different configurations and the continuation of fine-tuning experiments in the future. The setup includes the CLIP model and dataset loading, training loop, and evaluation loop. Aside from that, configurations for the models are different depending on the desired embedding layer size and the version of the CLIP model.

2.2.1 Model Loading

Challenges

There are a number of publicly available libraries or open source Github repositories for CLIP fine-tuning that can potentially be used as a starting point for the pipeline setup, since setting up the entire pipeline from scratch is a relatively time-consuming process, especially when we would be using multiple GPUs which involves the use of Distributed Data Parallel (DDP). We have reviewed and attempted to adapt from a number of currently available works, such as the transformers library of the HuggingFace platform and the source code from Dong et. al. [6]. However, these attempts were unsuccessful for the following reasons.

The HuggingFace transformers library was one of the first resources we looked at due to the fact that it provides classes and methods that automate many processes in the training pipeline, and it supported the use of multiple GPUs. Specifically, the `VisionTextDualEncoder` class from the transformers library is a model class that highly resembles the architecture of CLIP, with a vision encoder and a text encoder, and a contrastive objective. Another class `Trainer` essentially takes in a model under the transformers class and provides a `train()` method that automates the entire training process. Unfortunately, we were not able to utilize this library for two issues. Firstly, while it conveniently automated most of the steps in a complete training pipeline, there was a challenge in injecting custom behavior when the change needed involved overriding a series of methods in the `Trainer` class and the `Datasets`

class. Secondly, it was difficult to set up CLIP with the exact same vision and text encoder as the original paper using `VisionTextDualEncoder` and the available models. Although it offers many different vision and text encoder models, it would be best to keep the architecture of the CLIP consistent for meaningful comparisons with the original or a specific version of pre-trained CLIP.

Another pipeline set up by Dong et. al. [6] was able to load the original vision encoder of CLIP for training, but due to the objective of their research, which is fine-tuning solely for image classification tasks, the text encoder was not used and thus not loaded. As setting up the text encoder from scratch could be time-consuming and given the time available for the research, we decided to look further into other accessible implementations that have the objective of fine-tuning or pre-training the complete CLIP model.

Solution

Adapting from an open-source implementation of CLIP called `open_clip` led to the final success. It is a library that allows the loading of numerous versions of CLIP models and provides an implementation for pre-training experiments. Despite not being mentioned in its documentation, fine-tuning experiments are also possible given that it allows for loading the pre-trained weights to the model. In addition, new CLIP configuration files can be added to create a CLIP model with a customized embedding size, providing the ability to pre-train a CLIP with smaller embeddings. Lastly, it works flexibly with single and multiple GPUs which allows faster training. With this library, the basic model loading and training step and the use of DDP is automatically set up and only modifications in other steps and some additions are necessary.

2.2.2 Modifications

Modifications are made to inject custom behavior into data loading and the evaluation loop to best suit our data and hardware specifications. Additional statistics logging and checkpoint saving every n steps are also implemented to enable an easier recap

of the model's performance history and access to particular checkpoints. A new `CustomCLIP` class is created to enable a modified version of CLIP to be trained and loaded.

Dataset

For the `Dataset` class, images are loaded in two ways depending on the content of the CSV data files. Images for the train and test sets are stored locally and the path is constructed using the `item_id` field in the set. For all other datasets, images are retrieved through Fastly urls. Considering the size of our dataset, it is crucial to have a mechanism to handle corrupted images during the training in order to prevent abrupt discontinuation. While direct deletion of a corrupted sample is possible for single-GPU training, an error from the gather function for the distributed process would be raised as it expects the exact same number of items as the batch size for multi-GPU training. Our approach is to randomly retrieve another item from the entire set to replace the corrupted item. This indicates that the same item can potentially appear more than once per epoch, but such probability is low given the pre-filtering for corrupted images, and most experiments perform a random-crop transformation on the images each time they are retrieved, which makes the images vary each time. Lastly, for identification of the items when retrieving embeddings, `item_id` is also returned when an item is retrieved instead of just the text and image data.

Evaluation Loop

The performance metrics were originally computed after all the embeddings had been retrieved to obtain the global image-text pairs matching accuracy, which quickly led to out-of-memory error when the test set was not small enough, due to CLIP's large embedding size (512). To prevent this error, this accuracy is changed to be computed locally among n items in which n is small enough to prevent the memory error. A threshold argument is added to indicate the minimum number of items to compute the matching accuracy, and the metrics are computed for the accumulated chunk of items once the threshold is passed. At the end of the evaluation loop, a weighted

average of these chunk accuracies based on the number of items in each chunk will be computed and recorded. Details regarding evaluation metrics are discussed in section 2.4.

CustomCLIP

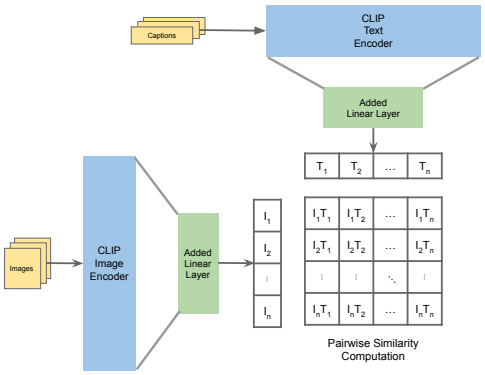

Figure 2-2: Architecture of the CustomCLIP class.

One of the fine-tuning experiments investigates the possibility of reducing the embedding size through adding a smaller layer to the original embedding layer of the image and text model. To enable the loading, training, and potential future usage of such a model, a new model class `CustomCLIP` is created with the architecture as shown in Fig. 2-2. The class takes in a CLIP model and has one additional linear layer to each of the encoders. For the convenience of fine-tuning solely on the last linear layers of the encoders when a fine-tuned CLIP checkpoint is already available, it has the option of locking the image and the text encoders of CLIP.

2.2.3 CLIP Training Flow

As shown and described in section 1.4, CLIP has an image-text dual encoder architecture with a contrastive objective. For all our experiments, the CLIP model

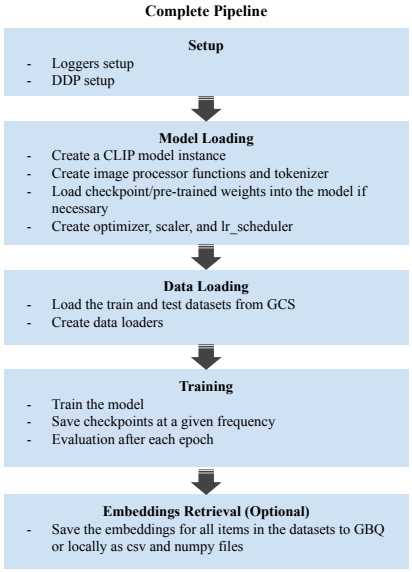

Figure 2-3: Overview of the complete training pipeline.

architecture is precisely as follows:

$$
\begin{aligned}
&\text{Dimensions: } D = 224,\ N \in \{128, 256, 512\},\ B \in \mathbb{Z},\ T = 77 \\
&x_{images} \in [0,1]^{B\times 3\times D\times D},\ x_{tokenized\ captions} \in \mathbb{Z}^{B\times T} \\
&\text{Items: } x = \{x_{images},\ x_{tokenized\ captions}\} \\
&\text{Image Encoder: } f_{Image\ Encoder} : [0,1]^{B\times 3\times D\times D} \mapsto [0,1]^{B\times N} \\
&\text{Text Encoder: } f_{Text\ Encoder} : \mathbb{Z}^{B\times T} \mapsto [0,1]^{B\times N} \\
&\text{Optimization Target: } \mathcal{L}_{InfoNCE}(f_{Image\ Encoder}, f_{Text\ Encoder})
\end{aligned}
\tag{2.1}
$$

in which D is the dimension size of an image, B is any arbitrary number of items (such as batch size), and T is the maximum number of words or tokens in a caption. The 3 in the dimensions of the images is the number of color channels for the images.

2.2.4 Complete Pipeline

Fig. 2-3 captures the complete process of a training experiment. An experiment begins with the setup of loggers to provide both clear progress logging in the terminal during training and performance metrics records that are updated and saved to GCS

along with model checkpoints. DDP is initiated to enable multi-GPU training. After the setup, a CLIP or CustomCLIP instance is created with the proper train and test image processor functions and the tokenizer. The train and test sets have different image processors because images need to undergo an extra random crop transformation for most of the experiments during training, whereas images in the test set are fed into the model without such transformation. Pre-trained or checkpoint weights are then loaded into the model if the experiment is a fine-tuning or continued experiment. To conclude this step, an optimizer, a scaler, and an lr_scheduler are created for updating model weights and the learning rate. With the model and necessary components placed, we load the train and test datasets from the csv files stored in a GCS bucket, and create the train and test dataloaders with the tokenizer and the corresponding image processor functions. The training finally begins at this point, with model weights updated at each step and an evaluation done after each epoch. Checkpoints and relevant performance statistics are also saved to GCS at a defined frequency during the training. Once the training complete, it can optionally save the text and image embeddings retrieved from the last checkpoint of the model, either to a given Google BigQuery (GBQ) table or locally as CSV and NumPy files.

2.2.5 Hardware Specifications

We need to consider the type and number of GPUs to use and where to set up an appropriate environment to run the experiments.

GPUs

We have experimented with several GPUs types on a Google Vertex AI workbench instance[2], including the Nvidia Tesla T4 16GB, Nvidia Tesla V100 16GB, and Nvidia A100 40GB. We decided to use the A100s due to its much higher memory than the priors, which allows for a larger batch size of 234 compared to a batch size of 70 for T4 and V100. A larger batch size is strongly preferred because of the learning objective of

[2]https://cloud.google.com/vertex-ai-workbench

CLIP. Intuitively, having to match a larger number of image-caption pairs motivates the model to learn more meaningful features, whereas matching a smaller number of image-caption pairs reduces the difficulty and inherently has a higher chance of getting a correct match. As such, the A100 has an absolute advantage over the other choices.

The number of GPUs can affect both the batch size and the training speed. As explained above, the batch size is equivalent to the number of image-caption pairs to match each step during training. In an all-gather scenario, in which outputs of the model from each GPU are first gathered and then used for computing training loss and model update, using n GPUs with a per-GPU batch size of b results in an $n \times b$ effective batch size. On the other hand, we can only consider a local loss, in which case the effective batch is equivalent to the per-GPU batch size, but two batches will be computed and update the model weights at a time, still speeding up the training. Provided the differences, we would like to compare which of the setting would lead to better results by using 2 A100 GPUs, with which we can also cut the training time by half.

Virtual Environment

We have come up with two options for creating a virtual environment to run the experiments: Google Vertex AI Workbench and Polyaxon[3]. Each of them has its advantages and disadvantages in the aspects of setup procedure, experiment management, and hardware resources.

Google Vertex AI workbench is a Jupyter notebook-based environment for running any data-oriented workflow. It offers a variety of machine and hardware types with intuitive configuration setup procedures, making it relatively time-efficient when we want to set up a new environment for an experiment. Aside from that, it has a simple integration with any Google services like Google Cloud Storage[4] (GCS) and Google BigQuery[5] (GBQ), both needed for retrieval or saving of data, model, and em-

[3]https://polyaxon.com/
[4]https://cloud.google.com/storage
[5]https://cloud.google.com/bigquery

beddings. In particular, a GCS bucket can be conveniently mounted to a workbench instance's notebook imitating a local file system to save files to or access files in the GCS bucket using GCS FUSE[6]. The only disadvantage is that we would need to set up multiple of the same instances in order to be able to run multiple experiments at the same time, provided that the hardware resources and the network bandwidth are fixed for each instance, making it impossible to run more than one experiment at a time given the resource requirements of our experiments. It is also more organized to limit to one experiment per instance as it is easier to track the CPU and GPU status and identify irregularities when only one experiment is involved. In sum, a Google workbench instance is time-efficient to set up and has seamless integration with many Google services, though instances need to be set up individually and experiments need to be more manually managed.

On the other hand, Polyaxon is a platform dedicated to machine learning workflows and model deployment. Its major benefit is that once the project and the environment file are set up, experiments, which are regarded as jobs, are automatically queued for execution upon creation. A job is created simply by a command with the necessary arguments in the command line after connecting to Polyaxon, without the need to repeatedly set up the environments to run multiple experiments. The libraries, connections, hardware specifications, and configurations specified in the environment file of the project would be automatically arranged by Polyaxon. Unfortunately, we decided not to employ Polyaxon for the scope and purpose of this research. Although it only requires a one-time setup, writing the environment files and verifying the setup is a non-trivial process. Considering the time available and the focus of the research, which does not involve model deployment, having a simpler setup and leaving more time to run the experiments would be preferable.

For the reasons above, we decided to utilize Google Workbench instances to conduct the experiments. All the instances have the same configurations as in Table 2.1 below and have the same environment setup.

[6]https://cloud.google.com/storage/docs/gcs-fuse
[7]https://cloud.google.com/compute/docs/gpus#a100-40gb

Machine Type	GPU	vCPUs	RAM	Data Disk
a2-highgpu-2g[7]	2 × NVIDIA A100s	24	170 GB	170 GB

Table 2.1: Workbench instances configurations.

2.3 Storage

Images of the train and test set, model checkpoints, and image embeddings of Mercari's items need to be kept in a way that is organized and easily accessible.

2.3.1 Image Data

As mentioned in Section 2.1.2, the network between the mounted Filestore database with images of the train and test set and the Google Workbench instance was a significant bottleneck during training, thus these images must be transferred to a local directory inside the notebooks to avoid the limited network bandwidth. We transferred them from the Filestore database to GCS so that we could utilize the multi-process feature provided by the `gsutil` tool[8] to more efficiently copy these images to all the created instances.

2.3.2 Checkpoints and Metrics

Likewise, model checkpoints and metrics logs are sent to the GCS bucket of the model by using the GCS python library. This makes the checkpoints and the statistics easy to be accessed from any Workbench instances for usage or examination and available to our team.

2.3.3 Embeddings

We can choose to save the embeddings to a provided GBQ table or locally in csv and numpy files for the list of item ids and their corresponding image features respectively. These options were provided because while GBQ allows a cleaner view of the data and supports direct SQL data manipulations, local csv and numpy files are more

[8]https://cloud.google.com/storage/docs/gsutil

efficient to work with for the tasks of this research, which includes constructing a searcher locally for performance comparison with Mercari's current system for brand and category prediction, to be elaborated in Section 2.4.2.

Vector Database

Item Embeddings can also be stored as the index of a searcher or a matching engine. A searcher or a matching engine is essentially a vector database with two major features: vector storage and nearest neighbor search, which offers a variety of applications. Furthermore, with optimized methods like ScaNN[11], embeddings are quantized into smaller sizes, making them more space-efficient.

We have considered two options for building such a vector database for evaluation and use case demonstration purposes, which are the publicly available ScaNN[9] python library and the Google Vertex AI Matching Engine[10]. Mercari has been using the Matching Engine in a number of applications due to the number of functionalities provided, including real-time index updates, filtered retrieval and index storage on the cloud, and its auto-deployment and auto-scaling ability for production use. However, we decide to employ the ScaNN library instead to construct the searchers locally given the scope of the research. The setup of a matching engine can be a time-consuming process because it involves setting up a VPC private network and has a large IP address pool requirement. As deploying and putting the fine-tuned or pre-trained model into production use is considered to be a future work to this research, the cloud-based model and index management features are not necessary. In addition, the ScaNN library already covers all functionalities needed for our purpose. In fact, the Matching Engine is built on top of the ScaNN library, which means they are essentially equivalent aside from the extra functionalities that Matching Engine supports. Therefore, similar to the training experiments, we can perform the evaluation by building and using the ScaNN searcher locally in a Google Workbench instance. With the setup of the notebook being trivial, merely ensuring sufficient

[9]https://github.com/google-research/google-research/tree/master/scann
[10]https://cloud.google.com/vertex-ai/docs/matching-engine/overview

space in data disk and RAM in the vCPUs, using ScaNN can greatly reduce the time needed to evaluate several models.

2.4 Evaluation

We evaluate the models in three different methods to better gauge their performance in both CLIP's original objective and classification tasks that are applicable to some of our use cases. For comparison among the base-line CLIP and the pre-trained or fine-tuned models, we evaluate on the image-caption matching tasks as well as a simple brand and category classification task. Upon selecting the best fine-tuned or pre-trained models, we compare the selected models with Mercari's current matching engine through brand and category prediction tasks using the ANN approach.

2.4.1 Model Comparison

During training, the models are evaluated with the test set on the image-caption matching task after each epoch. We compute the top-1, top-5, and top-10 accuracy of both image-to-text and text-to-image matching on each chunk of item images and captions, in which an image-to-text top-k accuracy means the percentage of items having their actual captions being predicted within the top-k predicted matching captions for their images. In mathematical representation:

$$
\begin{aligned}
f_{\textit{Image Encoder}} &: [0,1]^{B\times D\times D} \mapsto c_{image}, \quad c_{image} \in [0,1]^{B\times N} \\
f_{\textit{Text Encoder}} &: \mathbb{Z}^{B\times T} \mapsto c_{caption}, \quad c_{caption} \in [0,1]^{B\times N} \\
logits_{image} &= c_{image} \cdot c_{caption}^{T}, \quad preds_{image} = arg\,sort(logits_{image}) \\
logits_{caption} &= c_{caption} \cdot c_{image}^{T}, \quad preds_{caption} = arg\,sort(logits_{caption})
\end{aligned}
\tag{2.2}
$$

$$
\text{Top-k-Accr: } top\text{-}k_{image} = \frac{Count(actual_{image} \in preds_{image}[:k])}{B}
$$

Likewise for caption-to-image top-k accuracy computation.

in which $actual_{image}$ are indices of the actual captions for the images, and $arg\ sort$ is sorting in decreasing order. Additionally, we compute the mean and median ranks of the actual matching image or caption to measure how close the actual image and caption is toward being matched correctly. As described in Section 2.2.2, the chunk size is determined by a threshold passed in as the minimum number of items to compute the metrics over. We set this threshold to be 10,000 for all the experiments for not only consistency, but also the reason that 10,000 is large enough to prevent inflation in the accuracies and this number of image and caption embeddings could be accumulated for calculation without exceeding the memory limit based on preliminary training trials. After the training is complete, we determine the best epoch-checkpoint from the experiment by the highest top-1 image-to-text accuracy achieved since the most immediate use case of the model is brand and category prediction for the item by image. We will then compare the performances of the models' best checkpoints along with further evaluations. For this step of evaluation, we are using the test set with 72,810 items detailed in Section 2.1.

Following the selection of the models' best checkpoints, we perform an evaluation on both a top-50 brands and a top-100 categories classification task. Specifically, we gathered a data set of 14,086 items with the top-50 brands and a set of 30,442 items in the top-100 categories, with a category defined as the concatenation of the L0, L1, and L2 categories of the items with spaces in between. We first load the model's best checkpoint and create the same image processor and caption tokenizer as the ones used for the test set during training. The brands or categories are tokenized and encoded into embeddings. After that, we encode the images and compute their similarity scores with the embeddings of the brands or the categories. Similar to the prior evaluation, we calculate the top-1, 5, 10 accuracies and the mean and median ranks based on the computed similarity scores. This procedure is mostly identical to eq. (2.2), except that the number of captions are restricted to 50 and 100 for brands and categories respectively, and all images are computing the similarity scores $logits_{image}$ with these same captions. At this point, we will select several models with the best performances in brand and category predictions to be evaluated against

Mercari's current system, explained below.

2.4.2 Comparison with Current System

Mercari's brand and category auto-fill feature, or brand and category prediction by image, is powered by a matching engine as described in Section 2.3.3, with index built from the items' image embeddings retrieved from a pre-trained CLIP model. In order to truly understand how well the selected best models from the experiments perform in actual applications and how they compare with the current prediction system, we would like to build a ScaNN searcher for each of the best models to maximally replicate the approach used by the matching engine for prediction tasks. The complete operation can be defined as:

$$\begin{aligned}
\text{Items: } & x = \{x_{item\ id}, x_c, x_{brand}, x_{category}\} \\
\text{ScaNN: } & f_{ScaNN}(q, K) = \{x_i : \langle x_{ic}, q\rangle \leq \langle x_{i+1_c}, q\rangle\}_{i=1:K} \\
& q \in [0,1]^{B\times N},\ x_{ic} \in \mathbb{S}_{index},\ K, B \in \mathbb{Z},\ K \ll B \\
\text{Voting: } & Preds = Softmax(Count(x_{brand,category})) \\
& f_{voting} = arg\,max(Preds),\ \textit{top-k} = (arg\,sort(Preds))[0:k]
\end{aligned} \tag{2.3}$$

Precisely, for each query image q_j in the form of embeddings, we obtain its K nearest neighbors $x_{j,\{1:K\}}$, or most similar items, from the searcher, which we then use for a majority vote on the brand and category of the item in the input image. We first compute the top-1,5,10 accuracy like the previous evaluations. Aside from that, to enable comparison with the current system, we also compute a top-1 accuracy given a certain coverage. The matching engine only returns a prediction when the confidence of the prediction is beyond a certain threshold, thus for a given set of items to be predicted, the coverage is defined as the percentage of items that have received a

prediction result. Here is the illustration of this mechanism:

$$f_{matching\ engine}(q_j) = \begin{cases} arg\,max(Preds_j) & \text{if } max(Preds_j) \geq \theta \\ \varnothing & otherwise \end{cases} \quad \forall\, j \in B$$
$$Cov = \frac{Count(f_{matching\ engine}(q) \neq \varnothing)}{B} \tag{2.4}$$

To simulate this process, after we obtain the votes for the query images, we apply the softmax function to the vote counts and set a threshold θ of minimum softmax value to be considered a confident prediction. By tuning this threshold, we can maximally match the coverage to that of the results returned by the current system, and compare their top-1 accuracies while with similar coverage. For evaluation, we are using a 100,000 query set. In the aspect of searcher building, we are putting nearly 60 million items into the index of the searcher to match the minimum estimated number of items in the matching engine's index, as the process of embeddings retrieval from the model is a relatively slow process.

Chapter 3

Results and Discussion

This chapter details the configurations and evaluation of the trained models in this research. The first two sections talk about the pre-training and fine-tuning experiments respectively. Within each section, we provide the necessary hyper-parameters and data setup for the models, present their performance statistics, and discuss the possible implications behind these results. The last section presents the performance comparison of our two best models from the experiments with the currently used CLIP in brand and category predictions using the KNN majority-vote approach.

3.1 Pre-Training

We have completed 3 pre-training experiments in total, with the hyper-parameters and configurations shown below in table 3.1.

Experiment	pt_vitb16_5e-6	pt_embed128_5e-6	pt_vitb16_bc_only
CLIP Version	ViT-B/16	ViT-B/16-emb128	ViT-B/16
LR	5×10^{-6}	5×10^{-6}	5×10^{-6}
WD	0.1	0.1	0.1
Embed Size	512	128	512
Batch Size	234	234	234
Caption	brand_name name L2 category L1 category	brand_name name L2 category L1 category	brand_name L0 category L1 category L2 category

Table 3.1: Configurations overview for pre-training experiments.

We started with pre-training the original Open-AI ViT-B/16 CLIP (`pt_vitb16_5e-6`) as a baseline. Beyond that, the experiment `pt_emb128_5e-6` involved changing the

last layer of CLIP from a size of 512 to 128 to explore how much the performance would be compromised for smaller embedding size. Finally, we tuned on the caption's components by constructing the captions only with the brand and the category names, instead of including the *name* field, which was a user-filled title for the item. We hypothesized that with the caption focusing solely on the brand and category names, the resulting model would have a higher accuracy in specifically brand and category prediction tasks. We did not emphasize on tuning other numerical hyper-parameters because we believed that the data used for training would have a stronger impact on the learning results, whereas numerical hyper-parameters would more likely result in marginal improvement. The learning rate and weight decay values were chosen through preliminary experiments, through which we found that a smaller learning rate than that of the original pre-trained CLIP (5×10^{-4}) [42] was needed to prevent quick over-fitting due to our smaller dataset size.

3.1.1 Results

Figures 3-1 to 3-4 present the image-to-text and text-to-image top-{1,5,10} accuracies and mean and median ranks by epochs. The plots for the different models can have different lengths due to early stopping of the training once the plot appears to be converging. For the top-k accuracies plots, the solid, dashed, and dotted lines represent the top-1, 5, 10 accuracies correspondingly. In the mean and median ranks plots, the solid and dashed lines represent the mean and median ranks respectively.

At first glance, the first two models with captions including the item's *name* field are able to achieve better performances on the image-caption matching task than the model pre-trained on captions consisting solely of the brand and category names. Their top-k accuracies are around 5 to 10 percentage points higher and the mean and median ranks for the correct match are also higher by about 3 to 6. This lower accuracy for the matching task was expected for `pt_vitb16_bc_only`, since the task became more difficult as a result of the lack of variations in the captions for the same type of items. Suppose items A and B are the same objects, for `pt_vitb16_5e-6` and `pt_emb128_5e-6`, they can still have different captions as users most likely fill

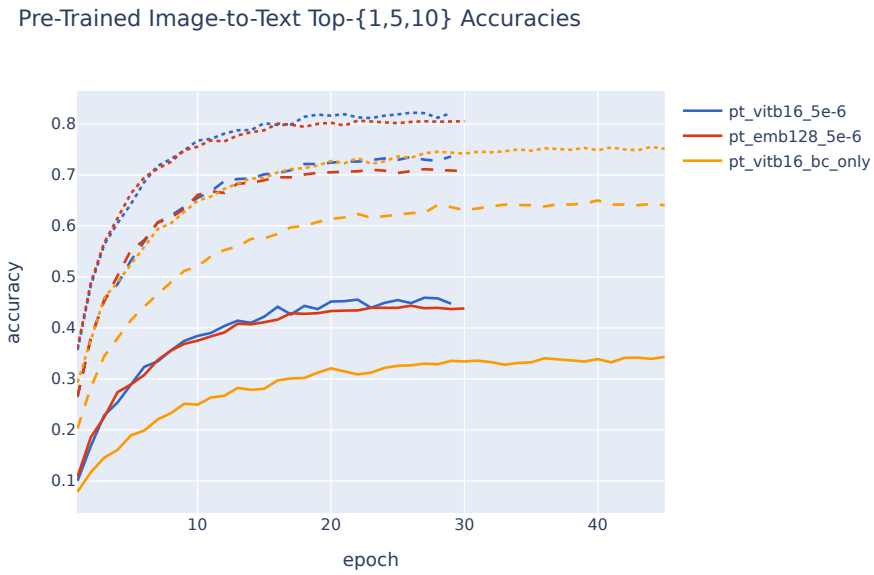

Figure 3-1: Image-to-text top-{1, 5, 10}-accuracies of the pre-trained experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively.

in *name* differently and potentially with some specific information unique to the items. However, A and B would have the same captions for `pt_vitb16_bc_only` since brands and category names are not text inputs but drop-down selections. For this reason, the task is more challenging when there are multiple same captions but the model must match the images to the exact captions assigned to them, in which $image_A$ must match to $caption_A$ but not $caption_B$ although $caption_A$ is equivalent to $caption_B$. Therefore, models' performances on classification tasks might better help us understand their abilities in recognizing and analyzing items from images, which we will provide later.

Comparing `pt_vitb16_5e-6` and `pt_emb128_5e-6`, the results were somewhat surprising. Although with the embedding size reduced to $\frac{1}{4}$ of the original size, `pt_emb128_5e-6` only suffered a decrease of no more than 3 percentage points in performance.

Table 3.2 shows the performance statistics for the best epoch of the models on the test dataset, along with the zero-shot of Open-AI ViT-B/16 as a baseline. For the matching task, both `pt_vitb16_5e-6` and `pt_emb128_5e-6` were able to out-

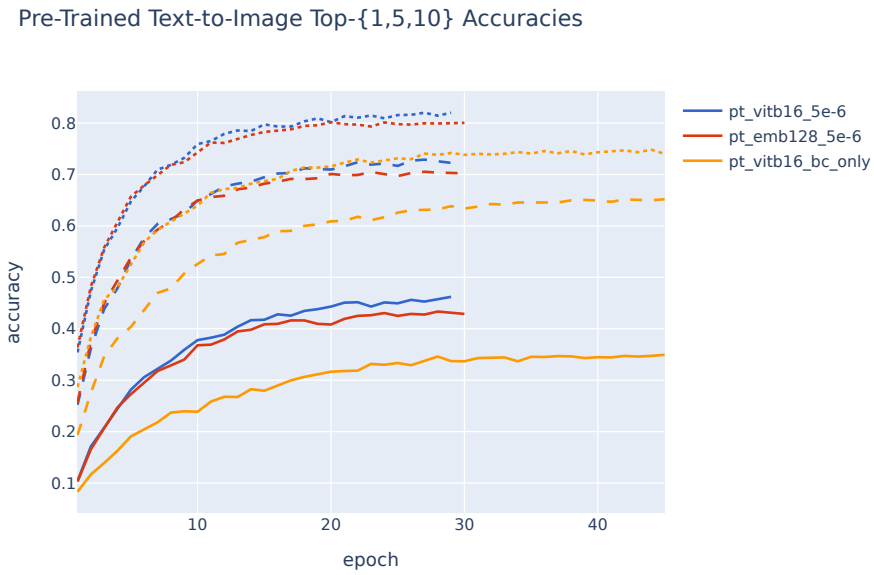

Figure 3-2: Text-to-image top-{1, 5, 10}-accuracies of the pre-trained experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively.

perform the zero-shot metrics, achieving 45.91%/44.29% and 44.36%/42.89% top-1 accuracy respectively compared to 37.75%/36.70% from zero-shot. The model `pt_vitb16_5e-6_bc_only`, however, was not able to reach the zero-shot performance, getting only 34.30%/34.93% top-1 accuracy at maximum. For brand and category classification, however, the performance of `pt_vitb16_5e-6_bc_only` becomes much more comparable to the other two pre-trained models, especially for category classi-

Task	Metrics	ViT-B/16 Zero-Shot	pt_vitb16_5e-6	pt_embed128_5e-6	pt_vitb16_bc_only
Image to Text	top-1	37.75%	**45.91%**	44.36%	34.30%
	top-5	60.02%	**72.98%**	70.71%	64.06%
	top-10	69.57%	**82.11%**	80.39%	75.08%
	mean	22.26	**17.47**	19.10	22.4
	median	3	2	2	3
Text to Image	top-1	36.70%	**45.29%**	42.89%	34.93%
	top-5	59.93%	**72.90%**	70.29%	65.19%
	top-10	69.74%	**82.03%**	79.71%	74.03%
	mean	23.02	**12.94**	15.29	18.21
	median	3	2	2	3
Top-50 Brands Classification	top-1	**40.28%**	25.45%	25.88%	23.56%
	top-5	**67.61%**	52.88%	51.95%	47.34%
	top-10	**78.40%**	68.17%	67.24%	62.50%
	mean	**6.79**	9.48	9.70	11.08
	median	**4.58**	5.74	5.93	7.29
Top-100 Categories Classification	top-1	**47.54%**	36.27%	32.66%	45.16%
	top-5	**84.37%**	71.19%	67.63%	80.52%
	top-10	**93.71%**	82.09%	79.03%	89.72%
	mean	**3.902**	8.01	8.57	4.74
	median	2.338	2.99	4.09	**1.92**

Table 3.2: Pre-Trained Models Best Epoch Performance Comparison

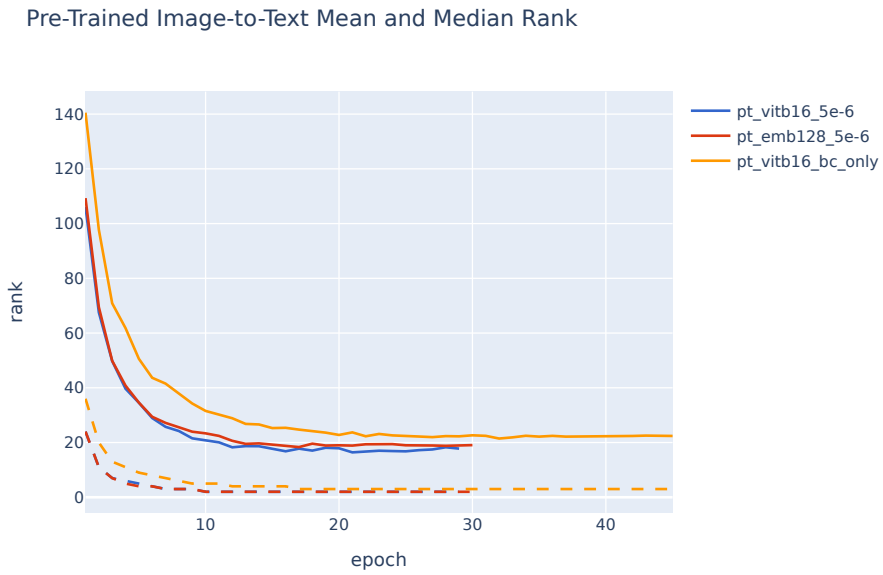

Figure 3-3: Image-to-text mean and median rank of the pre-trained experiments by epoch in solid lines and dashed lines respectively.

fication, in which it has its top-k accuracies almost 10 percentage points above the prior two. Although it still underperforms in brand prediction, the difference is much smaller than in the matching task.

Overall, two of the models pre-trained by Mercari's data were able to beat the zero-shot ViT-B/16 on the image-caption matching task, but the zero-shot CLIP still has the best performance in the classification tasks.

3.1.2 Discussion

The performance statistics of the pre-trained models have some potential implications on the effects of captions composition, embedding size, and dataset size.

In view of how the models pre-trained with captions consisting of different information perform differently in the tasks, the components of the captions definitely play a role in what features the model learns to seek and utilize from images. As we observed, models pre-trained on captions including free text inputs performed much better on the matching tasks whereas the model pre-trained on captions with strict

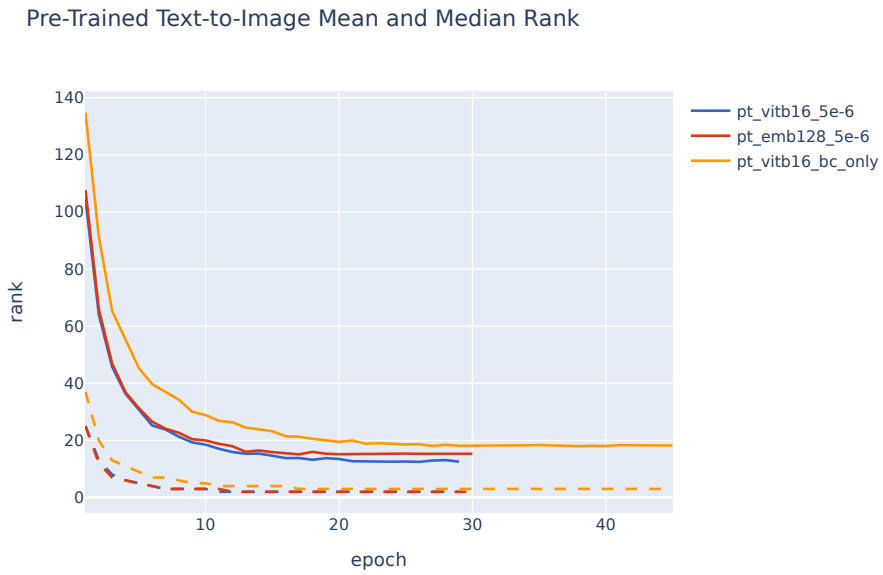

Figure 3-4: Text-to-image mean and median rank of the pre-trained experiments by epoch in solid lines and dashed lines respectively.

drop-down selection inputs could have better performance in the classification tasks. We do recognize that the latter still has a lower brand prediction performance, but this could be due to the fact that brands are inherently more difficult to be observed than categories, especially when there is nothing indicative on an item about the item's brand. In this case, the *name* field might offer more information that serves as extra guidance for brand prediction, enabling higher brand prediction accuracy for `pt_vitb16_5e-6` and `pt_emb128_5e-6`. In sum, the learning results from these pre-trained models suggest that while including natural language inputs as part of the captions allows the model to better examine pictures, identifying uniqueness and more subtle characteristics, using label-like captions helps the model improve in classification tasks due to the more focused learning of the possible labels.

Having tried different output-layer sizes for the pre-trained models, the relatively small performance difference between `pt_vitb16_5e-6` and `pt_emb128_5e-6` can have two interpretations. On one hand, the similar performances can be because of the insufficient data size, provided that CLIP is a relatively large model. The original CLIP model was trained on a 400M dataset, which is much bigger than our

dataset of size 7.3M. With this in mind, it is possible that our dataset is not large enough to drive the models to reach higher performance, regardless of the embedding size, and consequently the similar performances we obtained for the two models with the large embedding size difference. On the other hand, this might signify the potential of reducing the embedding dimensions without sacrificing significantly in performance, which means improved space-efficiency with little costs.

Finally, the failure of the pre-trained models to outperform the zero-shot CLIP in both the matching and the classification tasks is possibly because of the small dataset size compared to, as explained above. A larger dataset offers more variations in both images and captions, potentially allowing the model to become more robust even against image or captions patterns unseen before.

3.2 Fine-Tuning

We have conducted a total of 9 fine-tuning experiments, with configurations displayed in table 3.3. The experiments explored a number of factors that could perhaps affect the resulting performance, such as learning rate, caption composition, and image transformation. Considering the diverse setups of the experiments, here are the descriptions of each experiment for clarity:

- `vitb16_5e-6`: Fine-tuning the Open-AI ViT-B/16 CLIP with a learning rate of 5×10^{-6} and other configurations stated in table 3.3. The images used were pre-processed by resizing and center-cropping before training, and underwent a random-crop transformation and resizing during each epoch. The weights were updated with local loss described in section 2.2.5. This is the most basic setup and all the following experiments have the same setups except the parts specified in the descriptions.
- `vitb16_5e-5`: Fine-tuning with a learning rate of 5×10^{-5}.
- `vitb16_5e-6_n_b_f2`: Fine-tuning with captions constructed with a slightly different combination and order of the items' information.

- `vitb16_5e-6_resized_only`: Fine-tuning with images that were directly resized to 224×224, disregarding the original proportions, during pre-processing.
- `vitb16_laion2b`: Fine-tuning the laion2B-s34B-b88K version of ViT-B/16 CLIP.
- `vitb16_5e-6_add256`: Adding an additional layer to both encoders of CLIP and fine-tuning the customized CLIP. We loaded weights from the best checkpoint of `vitb16_5e-6` to the CLIP model, locked the image encoder weights, and then fine-tuned this custom CLIP model with captions comprised of brand and category names only. This design aims to preserve the model's knowledge regarding image feature extraction while letting the model learn deliberately on brand and category information extraction to potentially maximally preserve its performance in brand and category identification even with reduction in embedding size.
- `vitb16_5e-6_add256_name`: Having the same setup as `vitb16_5e-6_add256` except including the *name* field of the items in the captions.
- `vitb16_5e-6_norandcrop`: Fine-tuning without the random-cropping transformation during each epoch.
- `vitb16_5e-6_global_loss`: Fine-tuning with the same setup as `vitb16_5e-6`, but updating weights using global loss instead of local loss, described in section 2.2.5.

Similar to pre-training experiments, we focused more on tuning non-numerical components of the configurations as data-related modifications might produce a greater effect on the models' learning.

3.2.1 Results

The two-way matching top-{1,5,10} accuracies as well as the mean and median ranks by epoch for the image-caption matching task are plotted in figs. 3-5 to 3-8 for the models above. Like before, the plots for the model varied in lengths due to early stopping when convergence was observed. Almost all models followed similar learning trends, with relatively quick convergence even with a small learning rate. In summary,

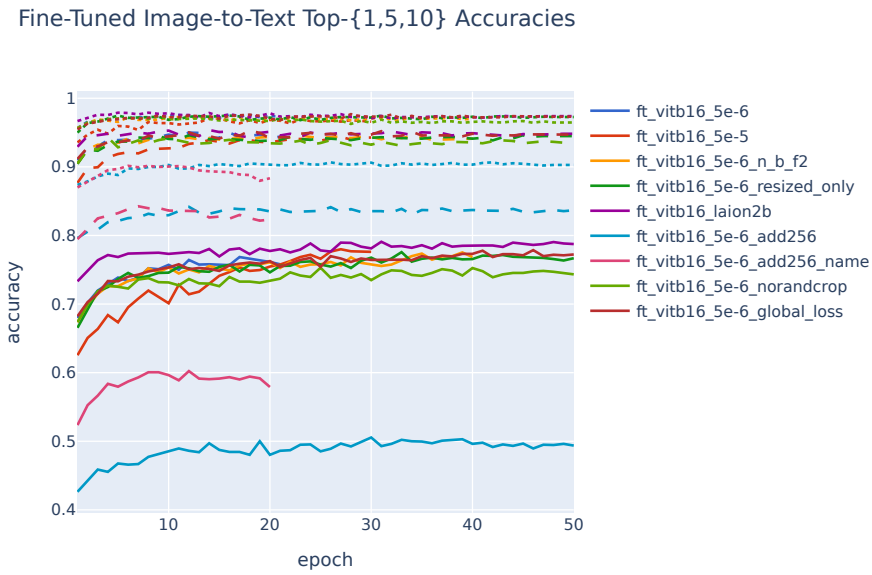

Figure 3-5: Image-to-Text Top-{1, 5, 10}-accuracies of the fine-tuning experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively.

most models are able to obtain a top-1, 5, 10 accuracy of around 75%, 94%, and 97% respectively, with a few exceptions. The mean rank lies between 2 and 3 for the majority, and the median rank is predominantly 1 except for one model.

The most noticeable difference among the models are the performances of the two models with a 256 embedding size compared to others, with much lower accuracies and higher mean and median ranks for the actual matches. The lowered performances are expected given a reduced embedding size implies less space for different features from the images or texts to be learned or represented. Another observation is that of the two models, the one learning from captions including the *name* field has better performance in the matching tasks than the one with captions consisting of brand and category names only, consistent to the discovery in the pre-training experiments.

For clearer comparisons, table 3.4 displays the performance statistics for the best epoch-checkpoint of each fine-tuned model, with the zero-shot CLIP performance as a baseline reference. The top-2 best values are bolded for most of the fields. Similar to the pre-training experiments, models who performed well in the matching tasks do not necessarily maintain the same performance during classification tasks.

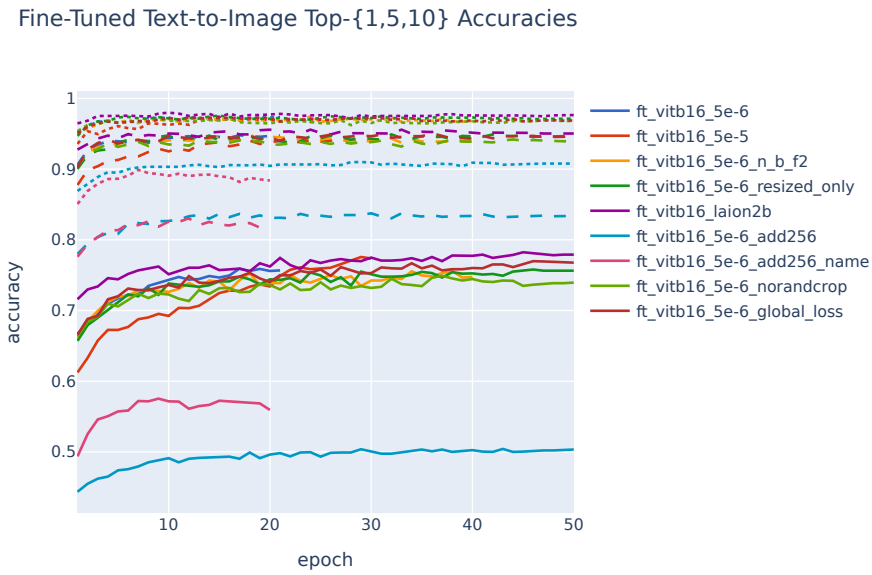

Figure 3-6: Text-to-Image Top-{1, 5, 10}-accuracies of the fine-tuning experiments by epoch, represented by solid lines, dashed lines, and dotted lines respectively.

For the image-caption matching tasks, `vitb16_laion2b` has the best overall performance, attaining 79.80%/77.06%, 94.91%/95.08%, and 97.43%/97.52% for top-1,5,10 both-way matching accuracies, to be compared with the zero-shot accuracies of 37.75%/36.70%, 60.02%/59.93%, and 69.57%/69.74%. The second best model appears to be `vitb16_5e-6` as it has the second most number of top values among the models, and it has its mean and median ranks closest to 1, which indicates that on average it has the best ability to bring the matching images and captions together. Its accuracies are 76.85%/75.93%, 94.36%/94.78%, and 97.39%/97.18%, not much lower than the prior one and still significantly outperforming the zero-shot CLIP.

Examining the classification performances, surprisingly, `vitb16_laion2b` was not able to maintain its best performances but in fact outperformed by a number of other fine-tuned models, whereas `vitb16_5e-6` was still able to reach a second-to-best performance with top-{1,5,10} accuracies of 67.14%/86.63%/92.90% and 57.17%/89.44%/94.71% for top-50 brands and top-100 categories prediction correspondingly. The best performing model, however, is interestingly `vitb16_5e-6_add256`, with brand and category prediction accuracies of 72.31%/91.03%/95.84% and 65.18%/93.53%/97.38%.

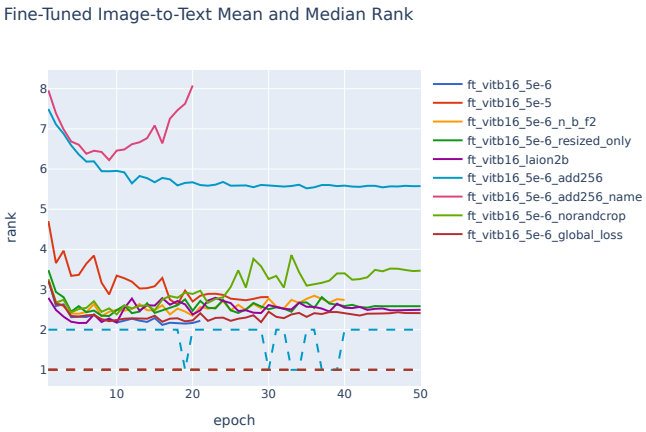

Figure 3-7: Image-to-Text mean and median rank of the fine-tuned experiments by epoch in solid lines and dashed lines respectively.

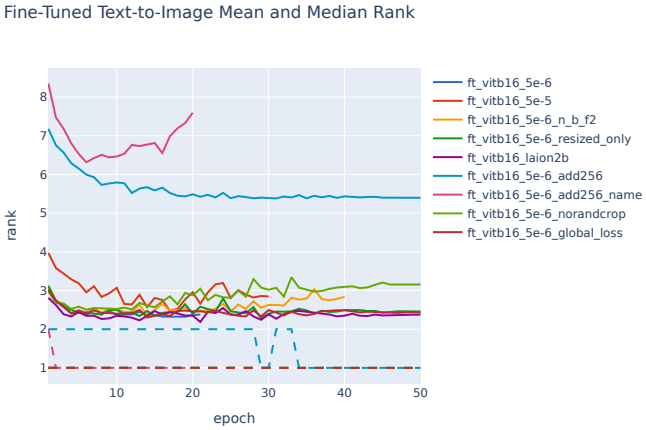

Figure 3-8: Text-to-Image mean and median rank of the fine-tuned experiments by epoch in solid lines and dashed lines respectively.

This is rather surprising at first glance, provided that its embedding is half the original size yet it is able to outshine the other model by approximately 5 percentage points on the top-1 accuracy. In view of the findings from the pre-training experiments, however, the performances of this model seem comparable to pt_vitb16_bc_only, to be elaborated in the discussion below.

Table 3.3: Configurations overview for fine-tuning experiments.

Experiment	ViT-B/16 Zero-Shot	vitb16_5e-6	vitb16_5e-5	vitb16_5e-6 _n_b_f2	vitb16_5e-6 _resized_only	vitb16_laion2b	vitb16_5e-6_add256	vitb16_5e-6 _add256_name	vitb16_5e-6 _norandcrop	vitb16_5e-6 _global_loss
CLIP Model	ViT-B/16	ViT-B/16	ViT-B/16	ViT-B/16	ViT-B/16	ViT-B/16	ViT-B/16	ViT-B/16	ViT-B/16	ViT-B/16
CLIP Version	openai	openai	openai	openai	openai	laion2b_s34b_b88k	ft_vitb16_ep17	ft_vitb16_ep17	openai	openai
LR	N/A	5×10^{-6}	5×10^{-5}	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}
Embed Size	512	512	512	512	512	512	256	256	512	512
Per-GPU Batch Size	N/A	234	234	234	234	234	234	234	234	234
Caption	N/A	brand_name name L2 category L1 category	brand_name name L2 category L1 category	name brand_name L2 category	brand_name name L2 category L1 category	brand_name name L2 category L1 category	brand_name L0 category L1 category L2 category	brand_name name L2 category L1 category	brand_name name L2 category L1 category	brand_name name L2 category L1 category

Table 3.4: Fine-Tuned Models Best Epoch Performance Comparison

Task	Metrics	ViT-B/16 Zero-Shot	vitb16_5e-6	vitb16_5e-5	vitb16_5e-6 _n_b_f2	vitb16_5e-6 _resized_only	vitb16_laion2b	vitb16_5e-6_add256	vitb16_5e-6 _add256_name	vitb16_5e-6 _norandcrop	vitb16_5e-6 _global_loss
Image to Text	top-1	37.75%	76.85%	**77.99%**	76.81%	77.57%	**79.80%**	50.55%	60.23%	75.08%	77.86%
	top-5	60.02%	**94.36%**	**94.36%**	94.19%	94.19%	**94.91%**	83.54%	83.54%	93.31%	94.74%
	top-10	69.57%	**97.39%**	96.59%	96.80%	97.14%	**97.43%**	90.61%	89.90%	96.63%	97.31%
	mean	22.3	**2.17**	2.73	2.64	**2.45**	2.57	5.59	6.62	3.17	2.41
	median	3	1	1	1	1	1	1	1	1	1
Text to Image	top-1	36.70%	75.93%	76.56%	74.83%	74.83%	**77.06%**	50.00%	56.10%	74.92%	**76.98%**
	top-5	59.93%	**94.78%**	94.19%	93.90%	94.40%	**95.08%**	83.75%	83.04%	93.77%	94.70%
	top-10	69.74%	97.18%	97.22%	97.10%	**97.26%**	**97.52%**	90.95%	89.02%	96.59%	96.84%
	mean	23.0	**2.32**	2.89	2.72	2.46	**2.27**	5.40	6.76	2.99	2.44
	median	3	1	1	1	1	1	1	1	1	1
Top-50 Brands Classification	top-1	40.28%	67.14%	**67.29%**	66.97%	66.31%	63.03%	**72.31%**	51.99%	65.81%	64.42%
	top-5	67.61%	**86.63%**	85.02%	86.30%	84.98%	84.63%	**91.03%**	75.57%	83.94%	84.85%
	top-10	78.40%	**92.90%**	90.93%	92.08%	91.55%	91.00%	**95.84%**	85.23%	90.35%	91.10%
	mean	6.79	**3.08**	3.72	3.28	3.49	3.64	**2.39**	5.07	3.79	3.63
	median	4.58	**1.15**	1.18	1.22	1.82	1.25	**1.04**	2.2	1.22	1.36
Top-100 Categories Classification	top-1	47.54%	**57.17%**	55.35%	50.26%	51.78%	48.22%	**65.18%**	46.35%	55.40%	49.09%
	top-5	84.37%	**89.44%**	84.83%	84.38%	84.50%	82.29%	**93.53%**	83.45%	86.61%	83.62%
	top-10	93.71%	**94.71%**	91.07%	91.87%	92.00%	92.26%	**97.38%**	92.00%	93.10%	91.40%
	mean	3.90	**3.21**	4.33	4.20	4.05	3.97	**2.31**	4.295	3.70	4.27
	median	2.34	1.69	1.93	1.96	1.85	2.09	**1.19**	1.94	**1.45**	1.78

3.2.2 Discussion

We will examine how each model performs compared to `vitb16_5e-6` to determine the possible effects the models' configurations bring.

`vitb16_5e-5`

This model has a fairly similar performance to `vitb16_5e-6`. One interesting observation is that it tends to have a better top-1 accuracy but worse top-5 and top-10 accuracy in most of the tasks compared to `vitb16_5e-6`. Also, it always has a larger mean rank compared to `vitb16_5e-6`, which possibly indicates that it is slightly less robust and on average has weaker ability to correctly match a pair than `vitb16_5e-6`. From these results, we infer that a smaller rate for a smaller dataset enables the model to attain a more stable or good average performance.

`vitb16_5e-6_n_b_f2`

We aimed to investigate whether the specificity of the components in the captions makes a difference in the training results with this fine-tuned model, for which we removed the L1 category name (L2 category is the most specific) from the caption components. In particular, we wondered if including less specific information in the caption would actually distract the learning due to their vagueness. The results do not diverge substantially from those of `vitb16_5e-6` for most of the tasks, but it does seem to consistently have lower performance than `vitb16_5e-6`. The difference is especially obvious for category classification, with accuracies about 3 to 7 percentage points below those of `vitb16_5e-6`, likely because of the lack of exposure to both the L0 and L1 category names during training. Therefore, these results possibly indicate that relevant information, although not necessarily precise enough, are safe and in fact necessary to be included into the data.

`vitb16_5e-6_resized_only`

Like the previous model, this model has comparable results for all but the last category prediction task to `vitb16_5e-6`. In addition, it has greater mean ranks for all the tasks, meaning it has a worse average performance. The results might suggest that preserving the items' proportion in the images are preferable to the model in order to learn better features of the items.

`vitb16_laion2b`

For the reason that the laion2b_s34b_b88k CLIP has a better overall performance than the Open-AI version, we tried to fine-tune this model to see if this superior performance could be retained. The results show that it is able to keep its performance in the matching tasks, having the highest top-k accuracies among the models. However, the performance dropped in the classification tasks, in which it is outperformed by a number of other models with lower image-caption matching accuracies. One possible reason might be that it becomes over-fitting to the caption compositions in the training and test set, which have the same format. Consequently, when it is given solely the brand or the category names as the captions, as in the prediction tasks, the difference in the captions limits its performance. Potential future study could be using an even smaller learning rate that can prevent over-fitting to see if the prediction performance can be improved.

`vitb16_5e-6_add256` **and** `vitb16_5e-6_add256_name`

While the two models have the same fine-tuning configurations except for caption compositions, the resulting performance seem to be quite different. In particular, `vitb16_5e-6_add256` exhibits identical behavior to `pt_vitb16_bc_only`, which suffers lower performance in the matching tasks due to the increased difficulty in the tasks explained in section 3.1.2, but excels in classification tasks because the captions it was fine-tuned with are more similar to the labels in these tasks. In contrast, `vitb16_5e-6_add256_name` has performance loss in all the tasks in response to the

reduced embedding size. These observations demonstrate that with embedding compressed to a smaller size, a more deliberate learning strategy is needed to retain the model's performance in the targeted tasks. Particularly, `vitb16_5e-6_add256` seems to have a good balance between general features learning and targeted learning. It first learned to adapt from general-context data (pre-trained ViT-B/16 CLIP) to Mercari data (`vitb16_5e-6`), after which the image encoder was locked to keep the learned knowledge for images and the model was further fine-tuned on the added linear layers as well as the text encoder to both compress the embedding size and learn to work with the new set of captions that are more similar to the labels of the prediction tasks.

`vitb16_5e-6_norandcrop`

This model consistently has a marginally lower performance than `vitb16_5e-6`, which signifies that the random crop operation to the images during each epoch is beneficial to the model's learning. This extra transformation allows the model to see slightly different images of the items each time, which possibly prevents over-fitting and makes it more robust against variations of the items' images.

`vitb16_5e-6_global_loss`

Having a doubled effective batch size for training, this model seems to be able to gain around 1 percentage point in the top-1 accuracy for the matching tasks compared to `vitb16_5e-6`. However, just as `vitb16_5e-5`, other performance statistics are mostly lowered. We suspect that same as how the learning rate affected the training of `vitb16_5e-5`, a larger batch size, while enabling faster features learning of the model, might lead to over-fitting and reduced robustness. With the quicker learning resulting from the batch size, it might have a better learning result if we employ a smaller learning rate to complement the larger steps in model weights updates during training.

3.3 Comparison with the ANN Approach

Viewing all the evaluation results in both the pre-training and fine-tuning experiments, we have chosen two models which we believe to have the best performances or most suitable for production use to be evaluated against the current CLIP model employed by Mercari's matching engine. We compare their performances in brand and category predictions with the ANN majority-vote approach for a set of 100K arbitrary items (query set) listed between 2023/05/10 and 2023/05/11. To construct a searcher for each of the models similar to the matching engine, we retrieved the image embeddings of a set of approximately 59.9M items to be put into the index of the searcher, and utilized the ScaNN python library to build the searcher as well as obtaining nearest neighbors for the query set. After that, we implemented the majority-vote classification approach to obtain the final predicted brands and categories for the items, which we used to compute the top-{1,5,10} accuracies and an additional top-1 accuracy taking into account the coverage factor explained in section 2.4.2.

We have selected `vitb16_5e-6` and `vitb16_5e-6_add256` based on the overall performances in the previous evaluation. The model `vitb16_5e-6` has one of the top performances and its performance is relatively consistent across all the tasks, which implies its strong adaptability to the different tasks. As for `vitb16_5e-6_add256`, although it seems to have a weaker ability for image-caption matching, it shows a outstanding performance in the prediction ability. More importantly, it has an embedding size that is half of the original CLIP and most other fine-tuned models, which makes it more space-efficient for embedding storage compared to other models, more suitable for production use. For these reasons, we would like to compare them with the current prediction system and investigate the possibility of adapting one of them for production use.

Task	Metrics	Current System	`vitb16_5e-6`	`vitb16_5e-6_add256`
ANN Brand	top-1	73.47%	**89.43%**	89.34%
	cov	47.92%	**49.93%**	49.89%
ANN Category	top-1	77.95%	**78.12%**	78.02%
	cov	64.66%	69.54%	**69.62%**
ANN Brand 100% Cov	top-1	N/A	**63.26%**	63.14%
	top-5	N/A	**83.57%**	83.19%
	top-10	N/A	**88.10%**	87.83%
ANN Category 100% Cov	top-1	N/A	**66.58%**	66.43%
	top-5	N/A	**90.76%**	90.64%
	top-10	N/A	**93.45%**	93.35%

Table 3.5: Brand and category prediction performance comparison with Mercari's current model and search system using the ANN majority-vote approach, maximally duplicating the system's prediction process.

3.3.1 Results

We have a total of 59,900,973 arbitrary items retrieved between 2022/10/01 and 2023/05/01 as the index set, which covers 19,845 brands and 3,343 categories. The calculated performance statistics are shown in table 3.5. To begin, the two selected models are equally competitive and have very similar performances, with `vitb16_5e-6_add256` no more than 1 percentage point lower than `vitb16_5e-6` for the accuracy values. They both out-shined the current system substantially for brand prediction, reaching 89.43% and 89.34% top-1 accuracy compared to the current system's 73.47%, all the while with higher coverage, meaning having more items receiving a prediction. The performances for category prediction are also marginally better, with slightly higher accuracies and an increase of about 5 percentage points in coverage.

When ignoring the coverage factor, which means counting all predictions regardless of the confidence of the predictions, `vitb16_5e-6` always has higher accuracy values compared to `vitb16_5e-6_add256`, but the differences are extremely minor, all within 0.5 percentage points.

3.3.2 Discussion

The results of the two models are promising that they show clear improvements in the prediction tasks when they are used with the same ANN majority-vote approach as the matching engine does with the current model. Aside from that, the fact that we cut the embedding size by half while still strongly retaining the performances is encouraging in that it leads us to the possibility of further reducing the embedding size for even better space-efficiency in the future.

Chapter 4

Conclusion & Future Work

4.1 Conclusion

In conclusion, we explored a variety of data modification, training setting, and hyper-parameters tuning strategies for pre-training and fine-tuning CLIP to improve CLIP's performance within Mercari's data domain. Through the experiments, we found that a small learning rate is preferable for small datasets to ensure robustness and prevent over-fitting. Similarly, larger batch size tends to cause over-fitting and other hyper-parameters need to be adjusted accordingly to limit the update step size. In terms of data pre-processing, better performances are obtained when the proportions of the items in the images are preserved and a random-cropping transformation is applied during each epoch. For caption composition, captions comprise of solely label-like selection inputs tend to more effectively improve the model's performance in classification tasks, whereas captions including free text inputs enable the model to capture more subtle characteristics of items in the images, better adapting to Mercari's data distribution reflected by the higher image-caption matching performance.

Overall, we managed to improve CLIP's performance in actual use cases. The pre-trained models were unfortunately unable to surpass the zero-shot CLIP likely because of the much smaller dataset size compared to that the original CLIP was pre-trained on. On the other hand, the best fine-tuned model achieved an 89.43% top-1 accuracy in brand prediction and a 78.12% top-1 accuracy in category prediction, compared to

73.47% (brand) and 77.95% (category) of the currently used CLIP model, all the while with higher prediction coverage. Additionally, we were able to reduce the embedding size of the best performing model from 512 to 256 while still firmly retaining its performance, attaining 89.34% and 78.02% for brand and category prediction with nearly identical coverage as the best model, which demonstrates the effectiveness of the fine-tuning strategy employed for this model.

4.2 Limitation and Future Work

Given the time restriction, not all aspects and possibilities concerning CLIP fine-tuning were examined or fully explored. We highlight some of the potential future work for this research below that either pushes for further optimization in the performance of the model or completes the model deployment and testing pipeline to enable production use of the best model obtained from this research.

4.2.1 Larger Dataset & Coverage

One direction to explore further is to pre-train and fine-tune CLIP with a much larger dataset. Pre-trained models from this research appears to suffer from the incapability of handling variations to the captions or unseen examples in general possibly because of over-fitting to the small training set. It would be interesting to see if a much larger dataset including more item variations allows for greater robustness against unseen items for both pre-training and fine-tuning cases, and the extent of such improvement.

4.2.2 Further Embedding Size Compression

It may be possible to further reduce the encoder embedding vector sizes to lower than 256 in view of the strongly preserved performance of the model after the embedding size was cut by half. This will further optimize storage efficiency and improve downstream task efficiency.

4.2.3 Production Conversion & Online Test

In this limited study, we have not truly executed all the intermediate steps to a commercialized production standard. Observing the results from this study, we can engineer the data processing and model training into an automated framework of choice, attaching with a periodic re-training schedule to produce updated model copies at frequent intervals. In addition, we can perform continuous deployment with automated online AB test experiments to track the performance differences at scale over the marketplace, where we can use additional online metrics to further verify the claims from the offline study here on a different perspective. This would also require us to further investigation optimizations needed for a deployed copy and how to efficiently serve the model on an endpoint, and use a production-ready ANN index to store and retrieve items, such as Google Vertex AI Matching Engine.

4.2.4 Comprehensive Hyper-Parameters Tuning

Due to time constraints we have under-explored the hyper-parameter space in this study. We can simply apply techniques such as Bayesian optimization [34, 21, 48, 20, 46, 3] here to obtain a more optimal model.

4.2.5 Comprehensive Image Pre-processing Techniques

In addition to resizing and cropping, we can also experiment with other image pre-processing tools[1], such as flipping, rotation, translation, contrast and more.

4.2.6 Additional Text Data Inclusion

In this study we have omitted a few available text data fields such as item description and other metadata. A potential improvement would be to investigate necessary cleaning and pre-processing steps to efficiently utilize them and include them in the text features.

[1]Many examples can be found at: https://pytorch.org/vision/stable/transforms.html

4.2.7 Parameter-Efficient Fine-tuning

While not explored in this study, there are a few more efficient ways to perform fine tuning on large neural network models. One of them is by using parameter efficient methods such as learning low rank adapter weight matrix updates like LoRA [13] in conjunction with lower floating point precision and quantization like QLoRA [4].

Bibliography

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikoł aj Bińkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: a visual language model for few-shot learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 23716–23736. Curran Associates, Inc., 2022.

[2] Arto, Sujit Pal, Mayank Bhaskar, Goutham, and Dev Vidhani. Fine tuning clip with remote sensing (satellite) images and captions.

[3] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010.

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[5] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models, 2022.

[6] Xiaoyi Dong, Jianmin Bao, Ting Zhang, Dongdong Chen, Shuyang Gu, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, and Nenghai Yu. Clip itself is a strong fine-tuner: Achieving 85.7accuracy with vit-b and vit-l on imagenet, 2022.

[7] Sepideh Esmaeilpour, Bing Liu, Eric Robertson, and Lei Shu. Zero-shot out-of-distribution detection based on the pre-trained model CLIP. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):6568–6576, jun 2022.

[8] Han Fang, Pengfei Xiong, Luhui Xu, and Yu Chen. Clip2video: Mastering video-text retrieval via image clip, 2021.

[9] Kevin Frans, L. B. Soros, and Olaf Witkowski. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders, 2021.

[10] Peng Gao, Shijie Geng, Renrui Zhang, Teli Ma, Rongyao Fang, Yongfeng Zhang, Hongsheng Li, and Yu Qiao. Clip-adapter: Better vision-language models with feature adapters, 2021.

[11] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020.

[12] Ziyu Guo, Renrui Zhang, Longtian Qiu, Xianzheng Ma, Xupeng Miao, Xuming He, and Bin Cui. Calip: Zero-shot enhancement of clip with parameter-free attention.

[13] 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.

[14] Chip Huyen. Real-time machine learning: challenges and solutions. *huyenchip.com*, Jan 2022.

[15] Masajiro Iwasaki. Pruned bi-directed k-nearest neighbor graph for proximity search. In Laurent Amsaleg, Michael E. Houle, and Erich Schubert, editors, *Similarity Search and Applications*, pages 20–33, Cham, 2016. Springer International Publishing.

[16] Masajiro Iwasaki and Daisuke Miyazaki. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *CoRR*, abs/1810.07355, 2018.

[17] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V. Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. *CoRR*, abs/2102.05918, 2021.

[18] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V. Le, Yunhsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision, 2021.

[19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[20] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21:345–383, 2001.

[21] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492, 1998.

[22] Jieh-Sheng Lee and Jieh Hsiang. Patentbert: Patent classification with fine-tuning a pre-trained bert model, 2019.

[23] Yoonho Lee, Annie S. Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts, Jun 2023.

[24] Ang Li, Allan Jabri, Armand Joulin, and Laurens van der Maaten. Learning visual n-grams from web data, 2017.

[25] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyun Ni, and Ning Wang. The design and implementation of a real time visual search system on jd e-commerce platform, 2019.

[26] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

[27] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023.

[28] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12888–12900. PMLR, 17–23 Jul 2022.

[29] Xianhang Li, Zeyu Wang, and Cihang Xie. An inverse scaling law for clip training, 2023.

[30] Haotian Liu, Kilho Son, Jianwei Yang, Ce Liu, Jianfeng Gao, Yong Jae Lee, and Chunyuan Li. Learning customized visual models with retrieval-augmented knowledge, 2023.

[31] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[32] Huaishao Luo, Lei Ji, Ming Zhong, Yang Chen, Wen Lei, Nan Duan, and Tianrui Li. Clip4clip: An empirical study of clip for end to end video clip retrieval, 2021.

[33] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[34] Jonas Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4:347–365, 1994.

[35] David Fraile Navarro, Mark Dras, and Shlomo Berkovsky. Few-shot fine-tuning SOTA summarization models for medical dialogues. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop*, pages 254–266, Hybrid: Seattle, Washington + Online, July 2022. Association for Computational Linguistics.

[36] Bolin Ni, Houwen Peng, Minghao Chen, Songyang Zhang, Gaofeng Meng, Jianlong Fu, Shiming Xiang, and Haibin Ling. Expanding language-image pretrained models for general video recognition, 2022.

[37] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. *CoRR*, abs/2112.10741, 2021.

[38] Daniil Pakhomov, Sanchit Hira, Narayani Wagle, Kemar E. Green, and Nassir Navab. Segmentation in style: Unsupervised semantic image segmentation with stylegan and clip, 2021.

[39] Rupa Patel and Anita Chaware. Transfer learning with fine-tuned mobilenetv2 for diabetic retinopathy. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–4, 2020.

[40] Matt Payne. 92.44pipeline for image similarity.

[41] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[42] 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.

[43] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, 2018.

[44] Hanoona Rasheed, Muhammad Uzair Khattak, Muhammad Maaz, Salman Khan, and Fahad Shahbaz Khan. Fine-tuned clip models are efficient video learners, Mar 2023.

[45] Aditya Sanghi, Hang Chu, Joseph G. Lambourne, Ye Wang, Chin-Yi Cheng, Marco Fumero, and Kamal Rahimi Malekshan. Clip-forge: Towards zero-shot text-to-shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18603–18613, June 2022.

[46] Michael James Sasena. *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations*. University of Michigan, 2002.

[47] Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, Andrija Antonijevic, Dax Pryce, David Kaczynski, Shane Williams, Siddarth Gollapudi, Varun Sivashankar, Neel Karia, Aditi Singh, Shikhar Jaiswal, Neelam Mahapatro, Philip Adams, and Bryan Tower.

[48] Simon Streltsov and Pirooz Vakili. A non-myopic utility function for statistical global optimization algorithms. *Journal of Global Optimization*, 14:283–298, 1999.

[49] Kohei Sugawara, Hayato Kobayashi, and Masajiro Iwasaki. On approximately searching for similar word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2265–2275, Berlin, Germany, August 2016. Association for Computational Linguistics.

[50] Mengmeng Wang, Jiazheng Xing, and Yong Liu. Actionclip: A new paradigm for video action recognition, 2021.

[51] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo-Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models, 2022.

[52] Ziyou Yan. Real-time machine learning for recommendations. *eugeneyan.com*, Jan 2021.

[53] Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan Li, Ce Liu, Mengchen Liu, Zicheng Liu, Yumao Lu, Yu Shi, Lijuan Wang, Jianfeng Wang, Bin Xiao, Zhen Xiao, Jianwei Yang, Michael Zeng, Luowei Zhou, and Pengchuan Zhang. Florence: A new foundation model for computer vision. *CoRR*, abs/2111.11432, 2021.