



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Text Augmentation from Extracted Attributes of Products in E-commerce using Vision-Language Model

Master Thesis

Study Program

Professional IT and Digitalization (Pro-ITD)

Faculty 4

Presented by

Shabbir Ahmad Farooquee

Date:

Berlin, 25.02.2024

1st Supervisor: Prof. Dr. Helena Mihaljevic

2nd Supervisor: Thomas Berger

Abstract

Multimodal deep learning models are being introduced to different practical applications and the e-commerce domain is one field where it is implemented for text generation of product images. Implementing such a system by traditional methods is complex in terms of creating, deploying, and maintaining because it consists of multiple models, where each model is trained with a task-specific dataset. Some large enterprises have implemented this with state-of-the-art model architecture which consists of a single model capable of extracting multiple attributes from product images to generate text. However, training these models from scratch requires millions of image-text pairs and huge computing power. The goal of this thesis is to present an alternative solution that can be implemented by small and medium-sized enterprises to augment text from product images. This is achieved by fine-tuning a pre-trained multimodal model for downstream tasks with custom but comparatively smaller datasets. The pre-trained model is selected based on its zero-shot classification capability to classify a product for multiple attributes. The base model is evaluated and custom datasets are created based on the attributes it is underperforming. The datasets are divided into two categories: primary and auxiliary. The purpose of an auxiliary dataset is to improve accuracy for an individual attribute while primary data is aimed to improve the text augmentation task. Evaluation of the fine-tuned model shows that the Top-1 accuracy has been increased by 3.81% and the Top-5 accuracy by 8.10%. This result indicates that model accuracy has been improved with fine-tuning and it can be improved further with a larger dataset and more training time.

Contents

Abstract	ii
List of Figures	iv
List of Tables	v
Chapter 1: Introduction	1
1.1 Motivation	1
1.2 Structure of the Thesis	2
1.3 Objective.....	3
Chapter 2: Background	4
2.1 Text Embedding.....	4
2.2 Image Embedding.....	7
2.3 Multimodal Embeddings	8
2.4 Multimodal Deep Learning	9
2.5 Contrastive Learning	9
2.6 Transfer Learning and Fine-tuning	10
Chapter 3: Related Work	12
3.1 Contrastive Language-Image Pre-Training (CLIP)	12
3.2 e-CLIP: Large-Scale Vision-Language Representation Learning in E-commerce	13
3.3 State-of-the-art solution	15
Chapter 4: Design	17
4.1 Business understanding	17
4.2 Data Understanding	18
4.3 Architecture for Text Augmentation	18
Chapter 5: Implementation	24
5.1 Data Preparation	26
5.2 Text augmentation for a product image	28
5.3 Fine-tuning	31
Chapter 6: Evaluation	33
6.1 Evaluation of Base Model	33
6.2 Evaluation of Checkpoint-1a	34
6.3 Evaluation of Checkpoint-2a	35
6.4 Evaluation of Checkpoint-1b.....	36
Chapter 7: Conclusion	37
References	38
Declaration of Independence	41

List of Figures

Figure 2. 1: Word embedding representations in a vector space.....	4
Figure 2. 2: word vectors with similar and different meanings	5
Figure 2. 3: Calculating the similarity between two vectors.....	6
Figure 2. 4: Text encoding used in language translation [22]	6
Figure 2. 5: Image clustering (similar images are clustered together).....	8
Figure 3. 1: Contrastive pre-training and zero-shot prediction using CLIP [8]	12
Figure 4. 1: Compare an image with a batch of texts [8]	19
Figure 4. 2: batch size for textual data is calculated for Medium-sized enterprise.....	20
Figure 4. 3: proposed architecture to reduce batch size	22
Figure 5. 1: Code snippet for calculating zero-shot weights for categories.....	24
Figure 5. 2: Code snippet for calculating zero-shot weights for categories (with templates).....	25
Figure 5. 3: A subset of image files and a snippet of the text file from the brand dataset	26
Figure 5. 4: A product in the dataset with the title “Nike T-Shirt Black”	27
Figure 5. 5: A subset of image files and a snippet of the text file from the image-title dataset	27
Figure 5. 6: Zero-shot weight calculation for all attributes.....	28
Figure 5. 7: Top-2 classification prediction for all three attributes	29
Figure 5. 8: A batch of title candidates is created from predicted classes	30
Figure 5. 9: Top-5 title match with probability	30
Figure 5. 10: Fine-tuning base model for two different paths.....	32

List of Tables

Table 5. 1: Comparison of zero-shot accuracy for all attributes with and without templates.	25
Table 6. 1: Zero-shot attribute classification (without template)	34
Table 6. 2: Zero-shot attribute classification (with templates).....	34
Table 6. 3: Evaluation result of Checkpoint-1a.....	35
Table 6. 4: Evaluation result of Checkpoint-2a.....	35
Table 6. 5: Comparison of Checkpoint-2a with the baseline	35
Table 6. 6: Evaluation result of Checkpoint-1b	36
Table 6. 7: Comparison of Checkpoint-1b with the baseline	36

Chapter 1: Introduction

1.1 Motivation

The success of using deep learning more than a decade ago in an ImageNet competition has inspired everyone to use neural networks for computer vision tasks resulting in fast progress in the field. Later, the introduction of transformer architecture [2] in the natural language processing field helped unprecedented progress in understanding natural languages and building large language models. The success of deep learning in computer vision and natural language has inspired researchers and companies to work on multimodal models (a combination of text, images, audio, and video).

Multimodal models today are capable of understanding and generating both images and texts to a certain extent that it has been introduced to various practical applications. E-commerce is one domain where the capabilities of the multimodal model are underutilized. Every product on the e-commerce platform has a set of images and texts (product title and description) associated with it. Every product information is uploaded as image-text pairs. For similar products, the text structure/format can be different if it is added by different sellers. This can be avoided and all products can have a consistent text structure (title and description) if the system is trained to generate texts based on attributes extracted from product images.

Training a multimodal model requires large datasets (millions of records as image-text pairs), lots of computing power, and human resources. Unfortunately, it is not feasible for SMEs (Small and medium-sized enterprises) to train a model from scratch because of the above constraints. To save time and resources, they can take a pre-trained model and further train with a small dataset specific to their environment using the concept of transfer learning and fine-tuning. Transfer learning and fine-tuning have proven very effective and efficient in different contexts in the field of computer vision and natural language.

One of the available pre-trained models is fine-tuned with custom data, specific to the requirement. The model is capable to extract attributes from a product image and use all extracted attributes to generate the product title. For training the model, the available image-text pairs from various e-commerce platforms are used. The texts in these cases require cleaning and editing to have consistency. The company can also use product information from

its archive. The model can be trained in such a way that it can associate images to texts where images can be used to extract attributes and compare them to all possible attributes for a match. Multiple attributes can be combined to augment text to form the title of the product. This method automates the task of extracting attributes from any product image and assigning or suggesting a product title. The process will bring consistency in the format of product titles across all the products. Also, it will bring accuracy and efficiency to this task.

1.2 Structure of the Thesis

Chapter 2 covers the background concepts to understand and implement this thesis work. It covers the summary of general concepts like text embedding, image embedding, and multimodal embedding to specific concepts like multimodal deep learning, contrastive learning, and fine-tuning.

Chapter 3 covers related work which is an inspiration for this work. It gives a summary of the CLIP model and research work done by NAVER Shopping. The chapter also covers the State-of-the-art solution for small-and-medium size e-commerce enterprises.

Chapter 4 introduces the design requirement for implementation, it covers the business understanding and data understanding sections. It also presents the architecture for the implementation.

Chapter 5 contains the implementation details, it talks about data preparation, text augmentation, and fine-tuning of a pre-trained model.

Chapter 6 has the record of each evaluation phase, which helps to compare the fine-tuning results with the baseline model.

Finally, Chapter 7 summarizes the achieved results and brings a conclusion to the work.

1.3 Objective

The main contributions of this thesis work are:

- 1) to understand the current solutions available to e-commerce enterprises in the context of multimodal deep learning.
- 2) extension of the solution to small and medium-sized enterprises which is only available to large companies.
- 3) to prepare custom data for fine-tuning a pre-trained model and have an implementation that can be achieved with fewer resources.
- 4) improvement in model performance after fine-tuning with custom dataset.

Chapter 2: Background

2.1 Text Embedding

Text embedding is a technique used in natural language processing that enables a computer to understand the semantics of words, sentences, or documents to represent them in a new form [1]. During this process texts are converted into numerical vectors which are a new representation of the input text. This is a crucial step in natural language processing since machine learning models cannot understand textual data in raw form and can only process numerical values. A text represented in the form of a vector can have an n-number of features where each feature carries some information related to the meaning of the text. Practically, a specific neural network (encoder) can be used to convert the text into an n-dimensional vector. The dimension is determined based on the number of words (text corpus) the model is trained on and can have somewhere between 8 to 1024 features based on small to large datasets. An example of word embeddings is shown in Figure 2.1; it has only two dimensions (features) for simplicity [22].

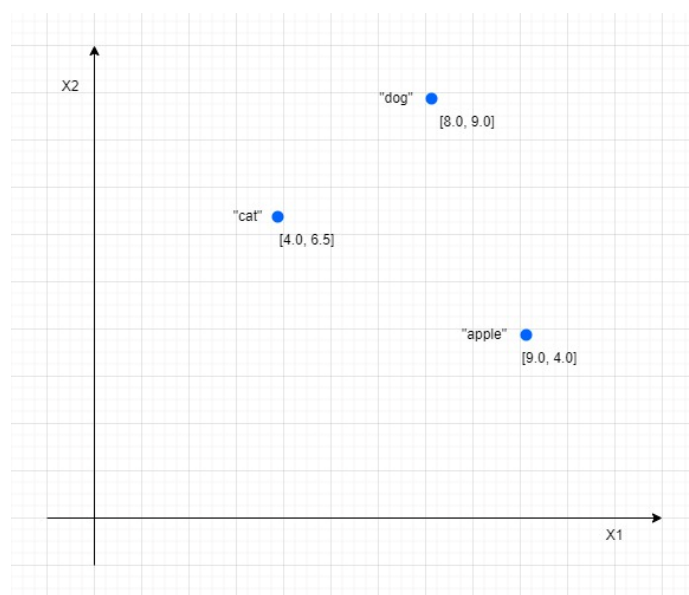


Figure 2. 1: Word embedding representations in a vector space

The same word used in two different contexts may have different meanings, in such cases, it will have different numerical representations based on context and will be separated in vector space. For example, the word “bank” has different meanings, one in the context of a river and another in the context of a financial institution. The different meanings in different contexts

result in different values for each feature in vector representation. This results in the same word being placed separately. On the other hand, two different words having similar meanings (synonyms) will have similar representations and will be drawn close to each other in vector space. The words “image” and “picture” are such examples (Figure 2.2).

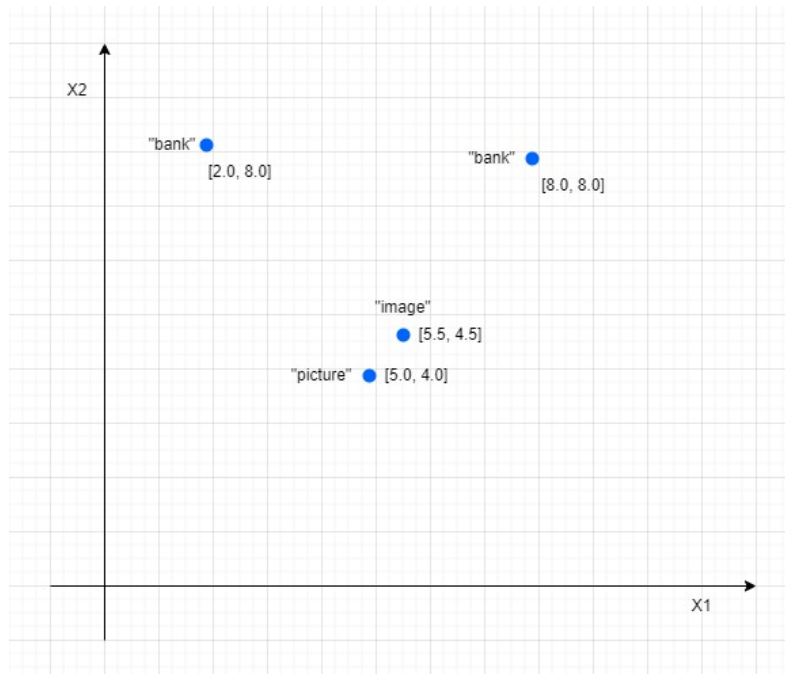


Figure 2. 2: word vectors with similar and different meanings

The word embedding supports arithmetic operations also, where arithmetic operations on two or more words can output a new word. A classic example of this is that the equation $\text{vector}(\text{"King"}) - \text{vector}(\text{"Man"}) + \text{vector}(\text{"Woman"}) \approx \text{vector}(\text{"Queen"})$ is satisfied if we replace all words with their respective numerical representation [1]. We can find similarities between texts by calculating the dot product between text vectors. For example, the similarity between two vectors (as shown in Figure 2.3) using dot product is calculated as:

$$\text{Similarity}(w1, w2) = w1 \cdot w2 = [4.0, 5.0] \cdot [7.0, 3.0] = 28 + 15 = 43$$

The other method to check similarity is from vector space by drawing vectors for each text representation and measuring the angle between vectors (θ). In this example, the similarity score would be calculated based on the value of $\cos\theta$ (as shown below in Figure 2.3).

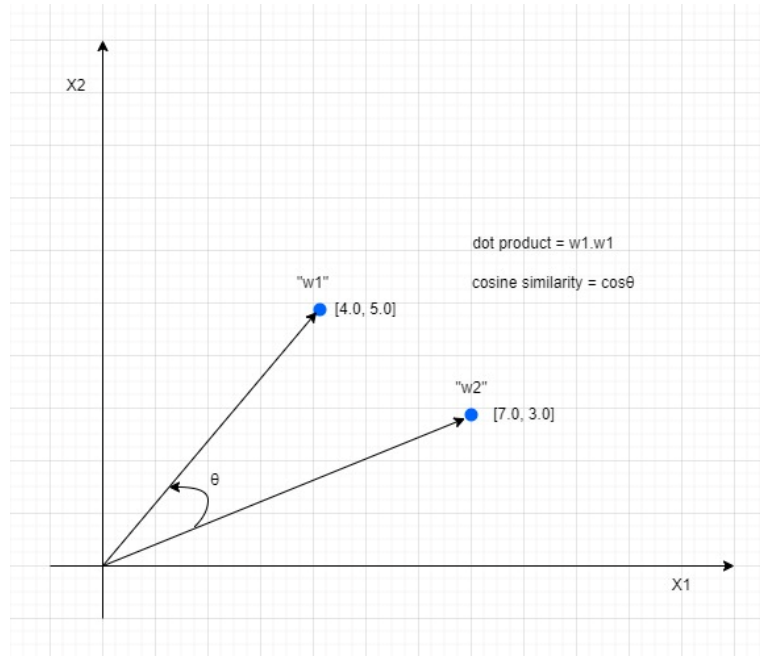


Figure 2. 3: Calculating the similarity between two vectors

As mentioned earlier, a specific neural network (encoder) is used to convert texts into vectors, and vectors can be converted back to text form by use of decoders. This concept is successfully used in many applications including language translation. In language translation, the encoder can take a variable-length sequence of words and convert it into a pre-determined fixed-length vector. In reverse operation, the decoder will take the vector representation of text and output a variable-length sequence of words as shown in Figure 2.4.

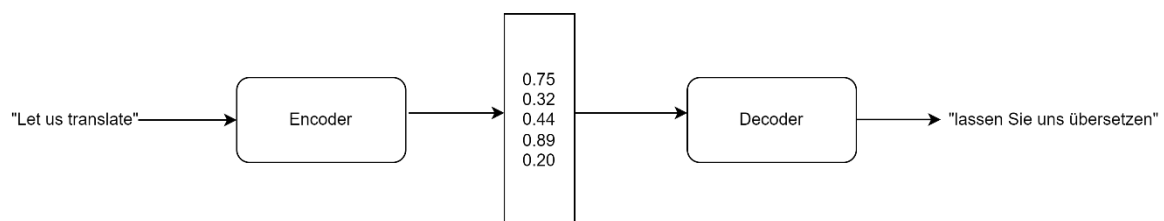


Figure 2. 4: Text encoding used in language translation [22]

Implementation of word2vec made a significant improvement in the representation of words semantically with simple model architecture compared to previous models. Because of the lower computational complexity, it became possible to process large text corpus (one trillion words) and compute accurate word vectors for high dimensional vector space [1]. The

efficiency of word vectors inspired the work for paragraph vectors, where a fixed-length vector representation is learned by an unsupervised framework for pieces of text. In this case, the model is trained to take variable length input of texts, such as sentences, paragraphs, and documents [3].

2.2 Image Embedding

As text embedding is the fundamental concept in natural language processing, image embedding is one of the core concepts in the field of computer vision. Originally digital-coloured images are 3-dimensional metrics where each element is considered a unit and called a pixel. So, each pixel represents the color of that particular unit in mathematical form. This raw form of the image can be converted into image embedding by a specific neural network. These neural networks are either CNN [4] or transformer-based models [5] which are trained with a very large number of labeled images. When images are converted into embeddings, they represent semantic and visual features of the image in numerical form and can be represented in a high dimensional vector space based on its feature values. Each feature in vector representation shows a particular attribute, for example, color, shape, size, etc. The vectors support arithmetic operations for images. For example, we can compare vectors for similarity check. Also, by changing the values of the features in the vector, we can change the color, shape, size, etc. Many applications, such as image classification, image labelling, clustering images, semantic image search, etc use image embedding. For example, to implement the classification of images, make a list of all possible classes and convert each class into embedding. Next, convert the input image into vector embedding, this vector can be compared for a match with all the class vectors extracted in the previous step. Another example is to use embedding space to cluster images as shown in Figure 2.5.

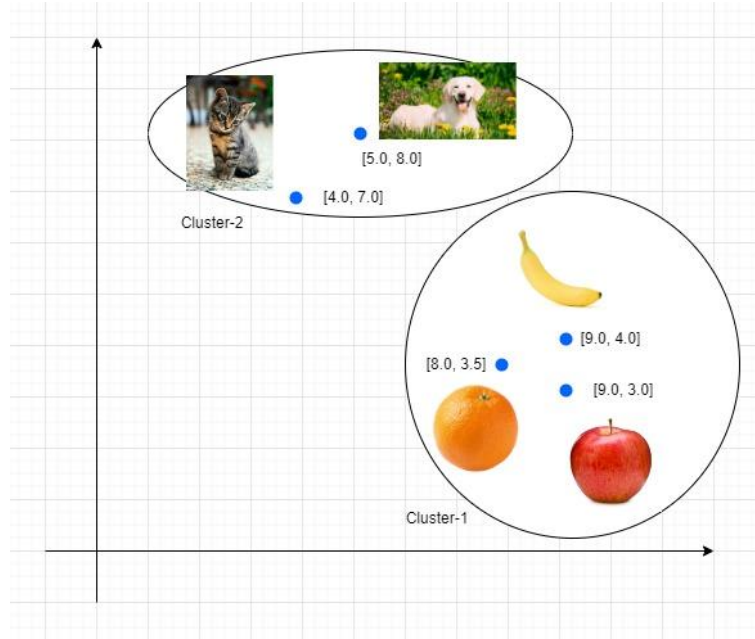


Figure 2. 5: Image clustering (similar images are clustered together)

2.3 Multimodal Embeddings

Texts and images in common embedding space are known as multimodal embeddings. It represents texts and images encoded in a common vector space of the same size. This is achieved by training computer vision models with a large dataset of image-text pairs. During the training phase, the model learns to understand the correlation between images and texts, and all images and texts are converted into a fixed-length vector to be represented in a common representation space. To perform conversion operations from images to vectors and texts to vectors, the models have separate encoders for images and texts [6]. Multimodal embeddings allow search for contents using input in one mode and list the search output in another mode. For example, even if images without annotation are saved in a vector database, they can be searched using textual string. The model can understand the semantic meaning of both textual and non-textual data [7]. To achieve this, all images are converted to vector form, then all vectors are saved and mapped with the corresponding image in a vector database. When a text is given as input to search for images, the text is converted into a vector, and the database is searched for similarity. For each match or image with a high similarity score, the images are displayed to the user in the original form. For example, when you type the text “shirt”, the encoder will convert it to a vector which is searched throughout the vector database for

similarity. After the search is complete, a list of shirt images will be displayed as a result. Another example could be the search for images based on a caption [6].

2.4 Multimodal Deep Learning

There has been unprecedented progress in the field of natural language processing since the advent of transformer architecture resulting in the development of many large language models. Large language models have shown intelligence but have uni-dimensional capability in terms of modalities. To expand the scope of language models, the multimodal deep learning concept is introduced. Multimodal deep learning is about neural networks that can consolidate multiple modes of information such as text, image, audio, and video [16]. These neural networks are trained on different modes of input data which is inspired by the way humans experience the environment using five basic senses. As a result, models can understand texts, images, audio, and video. A simple way to use a multimodal model is to give an image (for example: an apple) as input and ask it to describe it in the form of text [8] [9] or another way like sending a text as input “draw a picture of an apple” and ask the model to output an image. A multimodal model can also understand abstract concepts given in a text or shown in the images. It can describe a given image in textual form or it can draw an image based on textual input with an abstract concept [10]. Multimodal tasks usually handle unstructured data consisting of images and texts. It focuses on how to represent different inputs (images, text) numerically and then solves the problem of converting the numerical value to the desired form of either image or text [22].

2.5 Contrastive Learning

Supervised learning in computer vision has certain limitations, it is based on the training of the model with labelled images. These models depend on the quality of labelled data, which requires a lot of time to prepare and also needs expert knowledge of the data from specific domains. On the other hand, as part of self-supervised learning, a model based on contrastive learning learns from unlabelled data [22]. These models work based on a discriminative approach, converting images into vector embeddings without knowing the labels for each image. In vector space, it puts similar images closer to each other and separates different images. The same concept is used for texts also, where texts with similar semantics are plotted together and different semantics are separated. This process is further extended by the concept

of positive pairing and negative pairing, where the distance between the original image and augmented view of the same image (positive pairs) is minimized and the distance among embedding representations of different images is increased from each other. This process is enhanced by applying multiple augmentation methods, larger batch sizes, and extended training periods, which helps these unsupervised contrastive learning models [11].

2.6 Transfer Learning and Fine-tuning

Transfer learning is a concept in machine learning that is used to apply the knowledge learned from one task to solve other tasks that are different but related to the first task [12] [15]. Transfer learning has been used successfully in the field of research work and applied machine learning. Training a neural network from scratch for computer vision and natural language processing requires a large amount of training data, computing power, and training time (days, weeks, or more). Training a model for practical applications from scratch each time is not feasible and a pre-trained model can be used as a starting point for a specific application. A pre-trained model is a neural network that is trained with a large dataset and high computing power for one task and its weights are available to be used for further training with new datasets for another task. The knowledge learned by the model during pre-training is retained and it acquires new knowledge during further training. For example, a neural network to classify images consists of several hidden layers that can be divided into 3 categories, the first set of layers is used to detect edges, the second set of layers to identify shapes, and the last set of layers is used to represent image features and classification result. For further training of this model for a new classification task (with a set of new classes), only the output layer which shows the classification result from the previous task is removed and new layers are added instead. This way the new model has all the weights intact from the pre-training stage to identify edges, shapes, and image feature representation, and only the newly added layers are trained with the new dataset. This transfer learning process requires a smaller dataset, less computing power, and shorter training time because only the last few layers are updated at this stage.

Fine-tuning is a type of transfer learning. In contrast to the previous explanation where all the hidden layers are intact during transfer learning and only newly added layers are updated, fine-tuning allows some or all the layers to be updated during the transfer learning process. It can be decided during training what layers are frozen and what layers need updating. This decision can be taken based on the dataset used during pre-training and the dataset used during the

downstream task. Fine-tuning of pre-trained large language models and vision models for downstream tasks can be performed efficiently with low hardware resources and faster training time to update the model weights with datasets related to downstream tasks [13] [14].

Chapter 3: Related Work

3.1 Contrastive Language-Image Pre-Training (CLIP)

CLIP [8] is a multi-modal vision-language model trained on a variety of image-text pairs. It can be instructed in natural language to predict the most relevant text snippet, given an image, without directly optimizing for the task, similar to the zero-shot capabilities of GPT-2 [19] and GPT-3 [20]. CLIP is inspired by the work of Ang Li and his co-authors [17] for achieving zero-shot classification for various computer vision datasets using natural language supervision. Earlier, computer vision models could only classify images based on pre-determined object classes. In these classifiers, the meaning of labels is not retained and is simply converted into an integer losing its semantics. On the other hand, CLIP creates text embedding by extracting the features that retain the semantics. This allows CLIP to classify objects for new labels even before fine-tuning them with custom data. CLIP is a simplified version of ConVIRT [18] which makes it an efficient model to learn image representation from natural language supervision. It is trained from scratch with 400 million image-text pairs scraped from the internet. CLIP learns to represent images and texts in multimodal embedding space by parallelly training a vision encoder and a text encoder respectively. As a result, the cosine similarity of the image-text pair is maximized and cosine similarity between unpaired images and texts is minimized. This is achieved with a vision transformer [5] to extract visual features and a masked self-attention transformer to extract the text features as shown in Figure 3.1. The image encoder is also implemented with ResNet, making the image encoder consist of multiple variants [8].

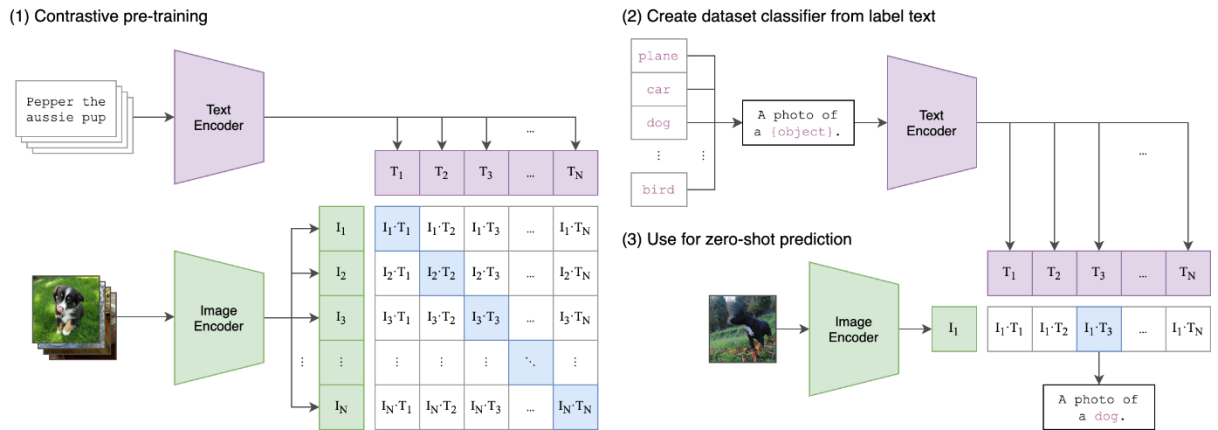


Figure 3. 1: Contrastive pre-training and zero-shot prediction using CLIP [8]

Images are sent to the vision transformer encoder as a sequence of patches; these patches are parts of the image which has fixed sizes and does not overlap with each other.

CLIP zero-shot capability on ImageNet matches the performance of the original ResNet50. This is achieved without pre-training with ImageNet original data which consists of 1.28M labelled examples. Additionally, CLIP's zero-shot performance is measured on over 30 different datasets for several tasks including fine-grained object classification, geo localization, action recognition, and OCR. It does not perform very well with fine-grained classification and object counting. For example, performance is very low when classifying different models of cars, and flower species or counting the number of objects in the image [8].

3.2 e-CLIP: Large-Scale Vision-Language Representation Learning in E-commerce

e-CLIP [21], a large-scale Vision-Language representation learning method for e-commerce is developed by researchers and engineers working at NAVER AI Research and NAVER Shopping. Naver Shopping is a shopping portal that has over 1.5 billion products with 500,000 sellers and affiliated malls. It has over 30 million weekly active users and over 2 billion monthly searches.

To provide access to millions of customers, the NAVER shopping portal supports various features, such as product search, price comparison, recommendations, and shopping content so that customers can access registered sellers on the portal. Naver shopping provides a few key functions: 1) product search based on product categories and attributes; 2) display a list of the same products from different sellers for price comparison.

NAVER shopping has developed a unified multimodal model that is symmetric for both modalities. However, implementing a large-scale multimodal pre-training model is non-trivial and puts several challenges: 1) the first and foremost challenge is a noisy dataset. The text part of many products contains irrelevant information and there is a significant number of pairs containing mismatched product texts with images; 2) the second challenge is a large percentage of the dataset contains duplicate products. This happens frequently when the same product is registered by different sellers; 3) the third challenge is inconsistency in product title format. The titles do not follow any grammatical structure, they are short and lack semantics; 4) the

fourth challenge is limited memory. Training a large model with a large batch size with limited memory of GPUs and TPUs puts several constraints.

NAVER shopping states that this is the first large-scale industry study investigating a unified multimodal transformer model that is symmetric for both modalities. They implemented an efficient and effective framework called e-CLIP which is based on contrastive learning. The e-CLIP framework utilizes a large-scale NAVER shopping dataset for the training and deployment of a unified model. It consists of three main parts: 1) the first part is data preparation which covers data collection and data preprocessing for one billion products; 2) the second part covers the modeling stage where the multimodal model is trained and evaluated with preprocessed data; 3) in the third and final stage, the pre-trained model is leveraged for downstream applications.

During the data preparation phase, the data is collected from the sellers which contains product images and descriptions including product title, brand name, price, etc. These details are stored in a distributed database after checking product details for compliance and policies. The data is taken from a distributed dataset for preprocessing and to make it ready for pretraining of the model. During preprocessing, the product images and texts are extracted as pairs from the complete dataset. In the next step, the quality of data is improved by filtering out three types of product data: 1) the invalid products are removed using a rule-based system for image-text pairs with no image, corrupted image, or very small image. Product titles are also checked for the number of words as acceptance criteria; 2) duplicate products are removed based on identical titles or duplicate images. Even though duplicate products are removed at this stage there are still a large number of duplicate products because of slight differences in product titles or images; 3) inappropriate products such as promotional social issues, immoral products, and adult products are removed based on compliance issues. During the process of data preprocessing, the dataset size is reduced from one billion to 330 million.

In the second phase, a multimodal model is pre-trained with the prepared image-text dataset. Inspired by the recent success in representation learning research such as CLIP [8] and ALIGN [9], NAVER shopping trained multimodal model based on contrastive loss where embedding representation of paired texts and images are similar, and embedding representation of non-paired images and texts are dissimilar. There are several advantages to these methods: 1) promising accuracy on downstream tasks; 2) having a symmetric framework where vision and

language models have independent encoders that can be used separately for modality-specific downstream tasks; 3) they can leverage vast amounts of unlabelled data.

For the original CLIP loss, image and text encoders are jointly trained using a symmetric cross-entropy loss by maximizing the cosine similarity of paired image and text embeddings while minimizing the similarity of non-paired embedding in a batch. This contrastive loss calculates each sample in the batch to be different from other samples. However, preprocessing could not remove duplicate data completely and these duplicate products are considered different products while training which are supposed to be treated as same products. This is problematic because the embedding of these same products is separated apart after calculating contrastive loss. This problem is solved by catalog-based soft labeling where contrastive loss is reduced by including catalog data. Despite limited computational resources, the model is trained with a large batch size by using the following techniques: 1) multi-stream accumulation; 2) learning rate to k-scheduler is replaced with batch size scheduled to increase model convergence; 3) mixed precision.

In the final stage of the system, the pre-trained model is used for these downstream applications: 1) category classification: it refers to classifying products into categories defined by NAVER shopping; 2) attribute extraction: identifying one or more attributes for each product; 3) clustering: grouping same products together which helps to identify new products; 4) product matching; 5) adult product recognition: to identify and filter out the products which are not appropriate for a minor. The model is evaluated for each downstream task. The results show that the e-CLIP model outperforms the baseline model for every downstream task and every evaluation scheme [21].

3.3 State-of-the-art solution

As mentioned in the previous section, NAVER Shopping implemented e-CLIP from scratch and has more than a billion image-text pairs in its database to prepare training dataset. The method adopted by NAVER Shopping will not be suitable for SMEs (Small and Medium-sized Enterprises) due to the following reasons: 1) they do not have a large database to prepare training datasets; 2) they may not have the high computing power to train large multimodal models; 3) to have a dedicated machine learning team may not be feasible. To overcome these

challenges, SMEs can embrace the concept of transfer learning and fine-tuning, which helps them to further train a pre-trained model with a smaller custom dataset available to them.

In this research work, we selected a pre-trained model that is suitable for extracting attributes from products on e-commerce platforms. The pre-trained model is evaluated as a baseline before further training. A custom set of images are collected from the e-commerce platforms to create image-text pairs for products. The texts in the dataset are defined based on the attributes included for forming the title of the product. The pre-trained model is fine-tuned with the created dataset and evaluated for the creation of a title with the extracted attributes from the image. The accuracy scores of the trained models are compared with the baseline to track the improvement.

Chapter 4: Design

4.1 Business understanding

E-commerce companies provide a platform for sellers and buyers to sell and buy products online. Sellers add their products to the platform where they upload images for each product and need to provide product information in textual form (title, summary, description, etc). Different sellers have different templates for product information, which results in inconsistency in product information format across the platform. This can be simplified if the platform can generate product titles from product images. In such a scenario, the sellers need to upload the image of the product and the platform will recommend the most suitable titles based on product attributes extracted from the image.

Currently, the product visual attributes are extracted by different models integrated with the platform. Each model is trained with a specific dataset and specialized to identify certain attributes. These models can only identify attributes it is trained on and only identify classes that are present in the dataset. For example, a model trained for color classification can only identify colors and not any other attribute. Also, it will only identify the colors it is trained on. There is another limitation that these models can only extract a single attribute but cannot generate text/title. Building such a system requires creating several models, where each model has a different architecture based on the attributes it is trained to recognize and requires a separate training dataset. After training each model separately, these will be integrated with the platform. Maintaining such a system will be tedious, and updating the models requires retraining them with new datasets.

OpenAI CLIP [8] simplifies the solution by integrating all the attribute extraction operations. The CLIP model is chosen based on: 1) zero-shot capability, it can identify product attributes even if it is not trained with a custom dataset; 2) it is one solution for all attributes, which means it is capable of extracting a diverse set of attribute list; 3) deploying and maintaining one model is much simpler and efficient as compared to previous method; 4) adding new values to existing attributes or adding a new attribute would be much simpler compared to previous methods.

4.2 Data Understanding

In this research work, we have selected three attributes: 1) category; 2) brand; and 3) color. The CLIP model will be evaluated to identify each attribute and then the model will be fine-tuned based on its result for each evaluation stage. We may need three datasets for fine-tuning and improving the evaluation score for each attribute: 1) a dataset containing image-category pairs; 2) a dataset containing image-brand pairs; and 3) a dataset containing image-color pairs. The high accuracy of extracted attributes is crucial for model performance because these attribute values are used to augment text for product titles. There will be one more dataset which is the primary dataset for fine-tuning. This consists of product image-title pairs and will be used for the final round of training. This dataset is divided into train-validation-test sets (70%-15%-15%).

4.3 Architecture for Text Augmentation

The CLIP model works with image-text pairs. To extract an attribute from an image, the image is required to be pre-processed and encoded by using an image encoder and then compared with a batch of embedded text as shown in Figure 4.1. To prepare a batch of encoded text, a list of all the possible values for each attribute is required. These values are passed to the CLIP tokenizer and text encoder as a single word or as a sentence as given in the below figure.

(2) Create dataset classifier from label text

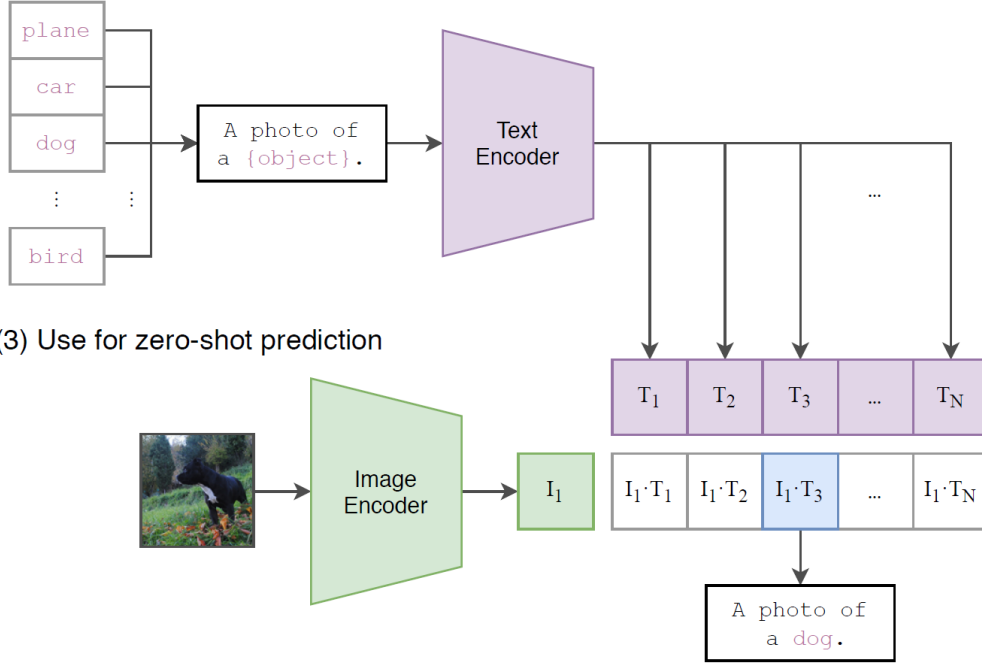


Figure 4. 1: Compare an image with a batch of texts [8]

Every product will have a list of attributes, which may vary from product to product. We have identified three attributes that are common for all products (as defined in the previous section). These three attributes are: 1) category; 2) brand; and 3) color. We can form a title for a product using these 3 attributes. For example, if brand = "Nike", category = "T-Shirt" and color = "Grey", then we can append all these attributes and form a product title = "Nike | T-Shirt | Grey".

For any enterprise, the administrator will form a separate list for these three attributes. This list can be small or big based on the enterprise size. For example, a medium-sized enterprise may have 100 brands, 50 categories, and 20 colors. The batch size for all the possible titles would be 100,000 as shown in Figure 4.2.

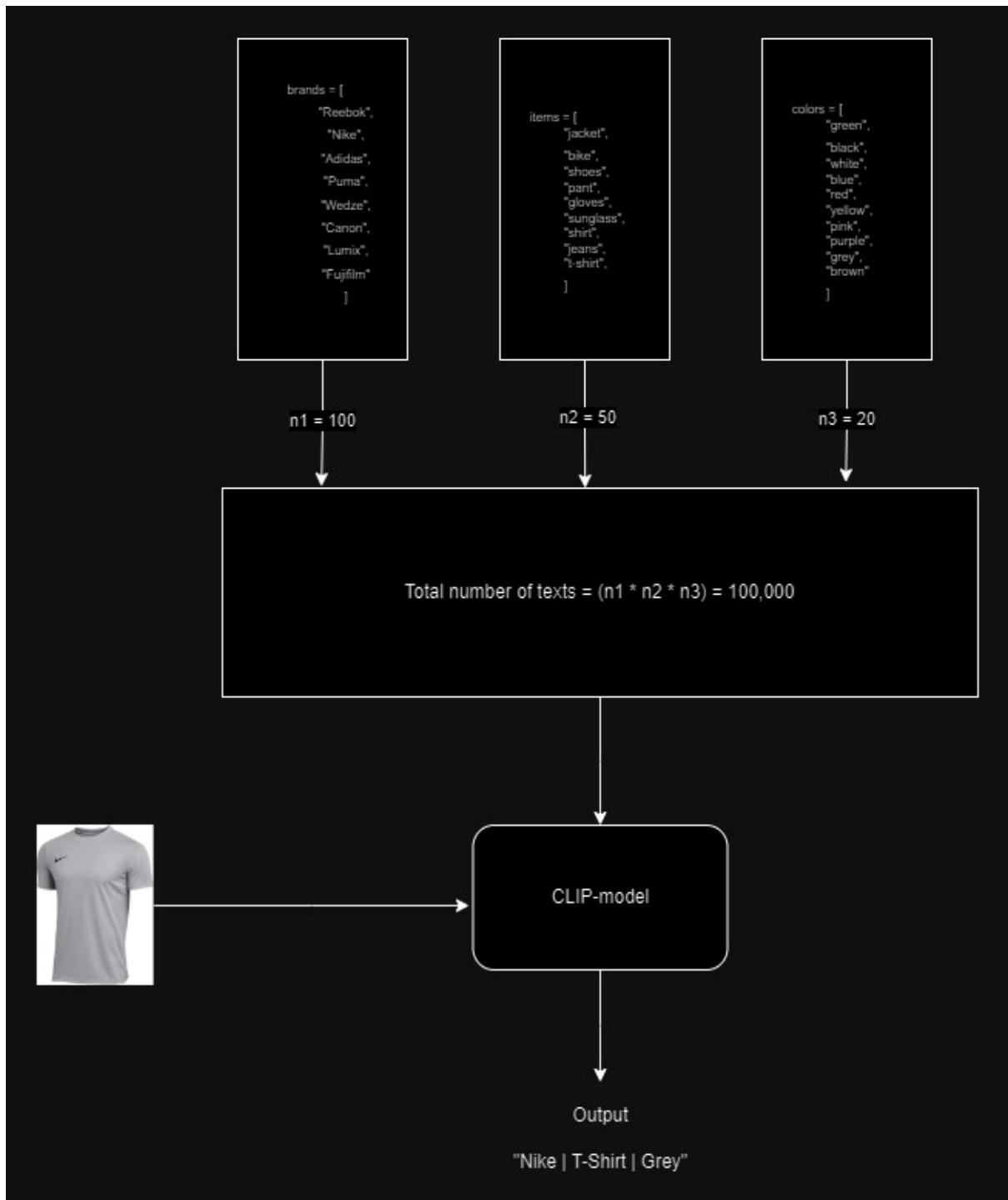


Figure 4. 2: batch size for textual data is calculated for Medium-sized enterprise

Next, these 100,000 titles will be converted to tokens and encoded by CLIP, and encoded values will be stored. Whenever any seller uploads an image, the CLIP encodes the image and compares it with all the encoded text for similarity and the best match will be selected as product title.

This architecture is not efficient because, for every product added, the image will be compared with 100,000 encoded texts. Further, if the size of the attribute list increases, then batch size also increases drastically and makes the system consume further computing power and much slower processing.

We propose an improved architecture to solve this bottleneck problem. Instead of forming a combination with all the values in each attribute, we process each attribute parallelly. In the context of the previous example, the 100 values of the brand are tokenized and encoded by CLIP and saved for future comparison. Similarly, 50 categories and 20 colors are also tokenized and encoded by CLIP and saved separately. For any image upload, the similarity score is calculated for each encoded brand value to select top-n brands. Similarly, the image will be parallelly compared with all category-encoded values and color-encoded values to select top-n categories and top-n colors (as shown in Figure 4.3). The default value of n is equal to 2 for each attribute. This architecture reduces the number of title candidates to 8, which is a drastic change from the previous value.

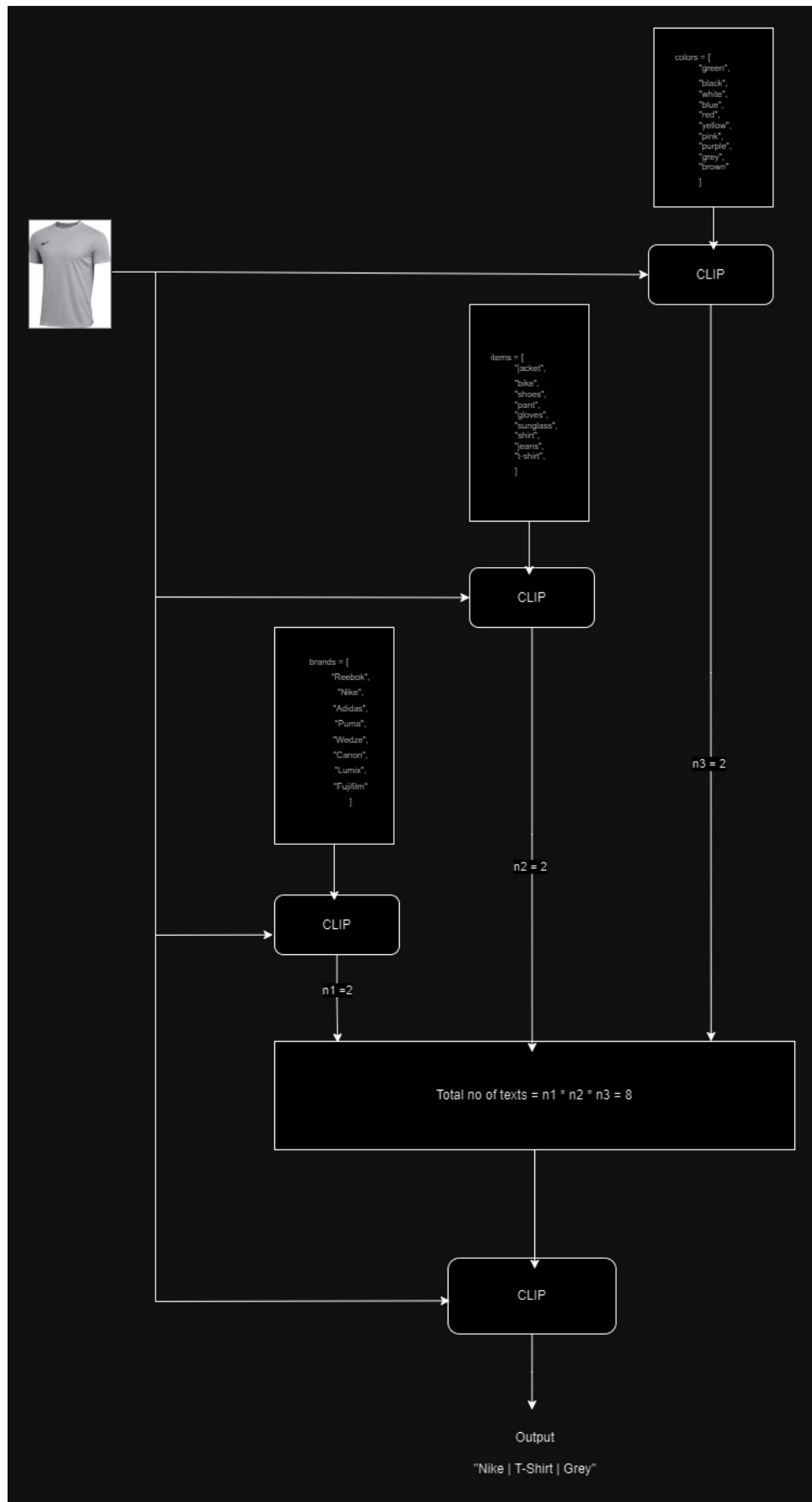


Figure 4. 3: proposed architecture to reduce batch size

As mentioned earlier, the default value is “ $n=2$ ”. But it can have any value “ $n \geq 2$ ”. It changes based on the performance of CLIP for that particular attribute and can have different values for different attributes. It means that the value of n for an attribute is independent of any other attribute. For example, if the default value of $n=2$ for the attributes brand, category, and color does not achieve accuracy $\geq 95\%$, then it is required to increase the value of n . If we achieve brand accuracy $\geq 95\%$ with top-3 brands, category accuracy $\geq 95\%$ with top-4 categories, and color accuracy $\geq 95\%$ with default value top-2 colors, then the values are $n=3$, $n=4$, and $n=2$ for brand, category, and color respectively. In this case, there will be a total of 24 title candidates which needs a similarity check with the product image.

Chapter 5: Implementation

For the implementation of this work, we have considered a small-sized enterprise, which has 10 categories, 40 brands, and 12 colors in its catalog. The values are the following:

```
categories = ["Sunglasses", "T-Shirt", "Shoes", "Jacket", "Socks", "Track Pant", "Shorts",  
"Cap", "Bag", "Beanie"]  
  
brands = ["Ray-Ban", "Carrera", "Gucci", "Versace", "Prada", "Tommy Hilfiger", "Lacoste",  
"U.S. Polo Assn.", "DKNY", "Polo Ralph Lauren", "Nike", "Adidas", "Puma", "Calvin Klein",  
"Reebok", "Under Armour", "Brooks Brothers", "Haimont", "ASICS", "Saucony", "FitVille",  
"Brooks", "Skechers", "Red Tape", "Little Donkey Andy", "33,000ft", "Columbia", "Carhartt",  
"MAGCOMSEN", "The North Face", "Darn Tough", "VRD", "G Gradual", "Fila",  
"BROKIG", "Champion", "NORTHYARD", "Mizuno", "Hurley", "Timberland"]  
  
colors = ["Black", "White", "Grey", "Brown", "Red", "Green", "Blue", "Orange", "Yellow",  
"Pink", "Violet", "Purple"]
```

We have evaluated the base model for three attributes and product title generation. For category evaluation, all the category values (single word) are tokenized and encoded as shown in Figure 5.1. These encoded values are saved and available for zero-shot classification of all product images in the test dataset. In this case, the embedded values tensor size is (512, 10).

```
In [ ]: categories = ["Sunglasses", "T-Shirt", "Shoes", "Jacket", "Socks", "Track Pant", "Shorts", "Cap", "Bag", "Beanie"]  
  
In [ ]: def zeroshot_classifier(attributes):  
    tokens = clip.tokenize(attributes)  
    class_embeddings = model.encode_text(tokens)  
    class_embeddings /= class_embeddings.norm(dim=-1, keepdim=True)  
    zeroshot_weights = class_embeddings.T  
  
    return zeroshot_weights  
  
zeroshot_category_weights = zeroshot_classifier(categories)
```

Figure 5. 1: Code snippet for calculating zero-shot weights for categories

The above method is tested with another variation where category values are placed in templates to form a sentence as shown in Figure 5.2. There are multiple sentences formed for each category value. After tokenizing and embedding sentences, the mean is calculated and a single embedding value is generated for each category. The final embedding size is the same as in the previous case (512, 10).

```
In [ ]: categories = ["Sunglasses", "T-Shirt", "Shoes", "Jacket", "Socks", "Track Pant", "Shorts", "Cap", "Bag", "Beanie"]

templates = [
    'the product is {}.',
    'the product is called {}.',
    'the item is identified as {}.',
    'the item is sold as {}.'
]

In [ ]: from tqdm.notebook import tqdm

def zeroshot_classifier(classnames, templates):
    with torch.no_grad():
        zeroshot_weights = []
        for classname in tqdm(classnames):
            texts = [template.format(classname) for template in templates]
            texts = clip.tokenize(texts)
            class_embeddings = model.encode_text(texts)
            class_embeddings /= class_embeddings.norm(dim=-1, keepdim=True)
            class_embedding = class_embeddings.mean(dim=0)
            class_embedding /= class_embedding.norm()
            zeroshot_weights.append(class_embedding)
        zeroshot_weights = torch.stack(zeroshot_weights, dim=1)
    return zeroshot_weights

zeroshot_category_weights = zeroshot_classifier(categories, templates)
```

Figure 5. 2: Code snippet for calculating zero-shot weights for categories (with templates)

Similarly, zero-shot weights for brand and color attributes are also calculated (with and without templates). The zero-shot weight tensors are (512, 40) and (512,12) in size for brands and colors respectively. The result for both cases (with and without a template) is available in the Evaluation chapter (Table 6.1 and Table 6.2). The two results are consolidated in Table 5.1 for comparison:

	Brand Classification		Category Classification		Color Classification		Title Augmentation	
	Without template	With templates	Without template	With templates	Without template	With templates	Without template	With templates
Top-1 accuracy	30.00	36.19	90.48	99.05	83.81	83.33	32.38	35.24
Top-2 accuracy	42.86	45.24	99.52	100	93.33	95.24	41.43	44.76
Top-3 accuracy	50.95	51.90	99.52	-	96.67	97.62	45.24	49.52
Top-4 accuracy	55.24	58.10	100	-	98.57	98.57	48.10	53.33
Top-5 accuracy	59.05	62.38	-	-	99.05	99.52	49.52	55.71

Table 5. 1: Comparison of zero-shot accuracy for all attributes with and without templates.

The comparison shows that the CLIP model performs slightly better for the classification of brands, categories, and colors with templates. So, we have decided to use templates for all further classification operations.

5.1 Data Preparation

The zero-shot classification for category and color attributes shows that CLIP is performing well with the default value of “n=2”. CLIP shows accuracy = 100% for the category attribute and accuracy = 95.24% for the color attribute with a default value of n. High accuracy means we do not need to increase the value of n. Also, we do not need a custom dataset for category and color attributes for fine-tuning CLIP, which is only required if CLIP is not performing well (accuracy < 95%). On the other hand, CLIP underperforms for brand attribute, the accuracy is 45.24% for the default value of n and it is 62.38% for top-5 (where n=5). Based on this outcome, we have decided that there will not be custom datasets for category and color attributes. We prepared primary and auxiliary datasets: 1) the auxiliary dataset is to improve brand classification accuracy, this dataset will be used for the first stage of fine-tuning the base model; 2) the primary dataset is to improve product title generation accuracy by fine-tuning the model with product image-title pairs.

Brand classification dataset for finetuning: This is the auxiliary dataset that covers all 40 brands. The images are hand-picked from the internet. There are 5 images for each brand, which makes a total of 200 images. The images contain the brand logo or brand trademark. The text file is a JSON file that is custom created based on picked images. It has the brand name in textual form for each image file, and the text is mapped with the corresponding image file location (as shown in Figure 5.3):

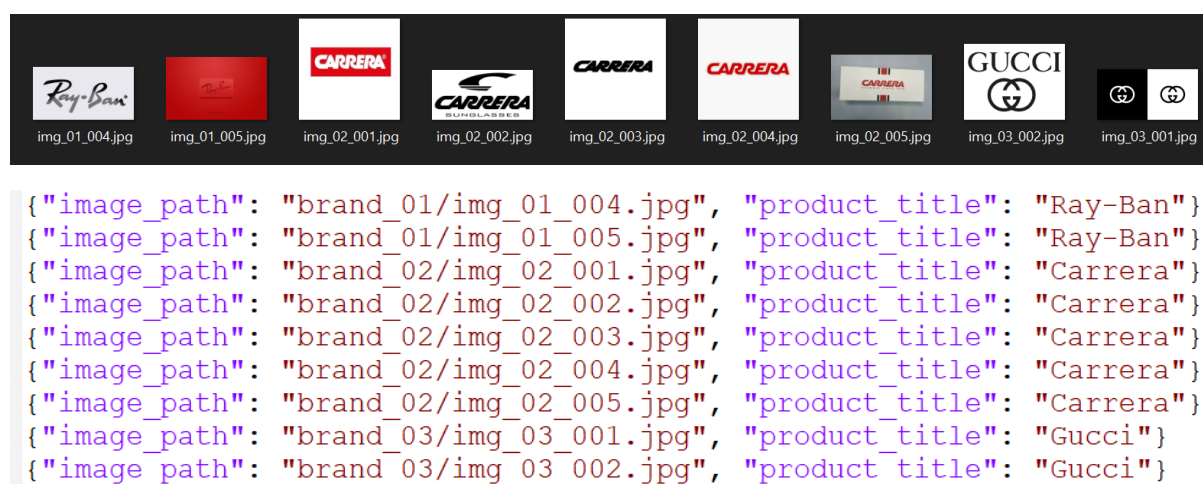


Figure 5. 3: A subset of image files and a snippet of the text file from the brand dataset

Image-title dataset for finetuning: This is the primary dataset and contains image-title pairs, where each image contains all three attributes and the title contains all attributes appended in a single text. For example, an image in Figure 5.4 represents category = T-Shirt, Brand = Nike, and Color = Black, so the title for this product is “Nike | T-Shirt | Black”.



Figure 5. 4: A product in the dataset with the title “Nike | T-Shirt | Black”

We decided to have a custom and balanced dataset for 10 categories, where each category has products from 10 brands. Each brand has 14 products under a selected category. This makes a total of “ $10 \times 10 \times 14 = 1400$ ” images. The whole dataset is balanced across categories, which means each category has 140 images. The dataset could not be balanced across brands, because some brands come under two or more categories. Also, it is very difficult to balance the data for color attributes because some items have a very limited color while others have a diverse set of colors. This dataset of size 1400 is divided into training-validation-test sets (70%-15%-15%), making the count of training=980, validation=210 and test=210. A subset of images and corresponding text is shown below (Figure 5.5):

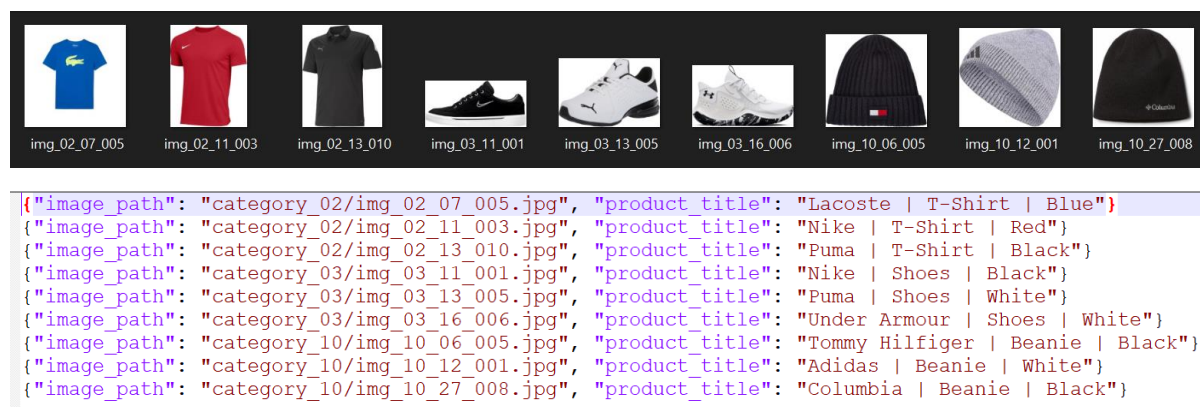


Figure 5. 5: A subset of image files and a snippet of the text file from the image-title dataset

5.2 Text augmentation for a product image

Text augmentation is the fundamental operation of this research work. This section explains how the CLIP model is used to augment texts and form a batch of product title candidates if a product image is given as input. The complete steps are implemented in Python as part of this work. This operation is divided into the following steps:

1) **Text processing and zero-shot weight calculation**: This step has lists of all attributes, which are integrated with corresponding templates. It is similar to the steps explained in the previous section, but in this case, all three attributes are taken, and each attribute has its own set of templates. After integrating lists with templates, texts are tokenized and encoded using the CLIP base model, and then weights are stored in three different variables as shown below:

```
from tqdm.notebook import tqdm

def zeroshot_weight_calculator(classification_list, templates):
    with torch.no_grad():
        weights = []
        for element in tqdm(classification_list):
            texts = [template.format(element) for template in templates]
            texts = clip.tokenize(texts)
            text_embeddings = model.encode_text(texts)
            text_embeddings /= text_embeddings.norm(dim=-1, keepdim=True)
            text_embedding = text_embeddings.mean(dim=0)
            text_embedding /= text_embedding.norm()
            weights.append(text_embedding)
        nn_weights = torch.stack(weights, dim=1)
    return nn_weights

zeroshot_brand_weights = zeroshot_weight_calculator(brand_list, brand_templates)
zeroshot_category_weights = zeroshot_weight_calculator(category_list, category_templates)
zeroshot_color_weights = zeroshot_weight_calculator(color_list, color_templates)
```

```
100% ██████████ 40/40 [00:22<00:00, 1.78it/s]
100% ██████████ 10/10 [00:05<00:00, 1.82it/s]
100% ██████████ 12/12 [00:06<00:00, 1.77it/s]
```

Figure 5. 6: Zero-shot weight calculation for all attributes

2) **Input image processing**: In this step, an input image is pre-processed and encoded using the CLIP model. The output is an encoded tensor of size [1, 512].

3) **Calculate cosine similarity and classification**: This step compares the image embedding with text embeddings for each feature. It finds Top-n predicted values where n has default value=2. The steps and output are shown in Figure 5.7.

```
brand_logits = image_features @ zeroshot_brand_weights
category_logits = image_features @ zeroshot_category_weights
color_logits = image_features @ zeroshot_color_weights

brand_probabilities = (100.0 * brand_logits).softmax(dim=-1)
category_probabilities = (100.0 * category_logits).softmax(dim=-1)
color_probabilities = (100.0 * color_logits).softmax(dim=-1)

top_brand_probs, top_brand_indexes = brand_probabilities.cpu().topk(2, dim=-1)
top_category_probs, top_category_indexes = category_probabilities.cpu().topk(2, dim=-1)
top_color_probs, top_color_indexes = color_probabilities.cpu().topk(2, dim=-1)

predicted_brands = top_elemsets(brand_list, top_brand_indexes)
predicted_categories = top_elemsets(category_list, top_category_indexes)
predicted_colors = top_elemsets(color_list, top_color_indexes)

print(predicted_brands)
print(predicted_categories)
print(predicted_colors)

['Nike', 'Puma']
['T-Shirt', 'Shorts']
['Red', 'Orange']
```

Figure 5. 7: Top-2 classification prediction for all three attributes

4) **Text augmentation from predicted classes**: The previous step gives top-2 brands, top-2 categories, and top-2 colors. We augment the title from these attribute classes and form a batch of 8 product titles.

```

predicted_titles = []
i=j=k=0

for i in range(len(predicted_brands)):
    for j in range(len(predicted_categories)):
        for k in range(len(predicted_colors)):
            predicted_titles.append(predicted_brands[i] + " | " + predicted_categories[j] + " | " + predicted_colors[k])

predicted_titles

['Nike | T-Shirt | Red',
 'Nike | T-Shirt | Orange',
 'Nike | Shorts | Red',
 'Nike | Shorts | Orange',
 'Puma | T-Shirt | Red',
 'Puma | T-Shirt | Orange',
 'Puma | Shorts | Red',
 'Puma | Shorts | Orange']

```

Figure 5. 8: A batch of title candidates is created from predicted classes

5) **Calculate probability for Top-n titles**: Text tokenization and text embedding steps are repeated for these title candidates, which forms 8 embedded texts to be compared with input image embedding. After comparison, the Top-5 titles are selected and the probability is calculated for each title as shown below (Figure 5.9):

```

probabilities = (100.0 * image_features @ predicted_title_encoded.T).softmax(dim=-1)
print("probabilities: ")
for p in probabilities[0]:
    print(p*100)

probabilities:
tensor(91.8496, grad_fn=<MulBackward0>)
tensor(0.5887, grad_fn=<MulBackward0>)
tensor(0.4779, grad_fn=<MulBackward0>)
tensor(0.0039, grad_fn=<MulBackward0>)
tensor(6.9767, grad_fn=<MulBackward0>)
tensor(0.0731, grad_fn=<MulBackward0>)
tensor(0.0298, grad_fn=<MulBackward0>)
tensor(0.0004, grad_fn=<MulBackward0>)

top_probability, top_index = probabilities.cpu().topk(5, dim=-1)

print("top probability: ", top_probability)
print("top index: ", top_index)

top probability: tensor([[9.1850e-01, 6.9767e-02, 5.8866e-03, 4.7786e-03, 7.3067e-04]],
 grad_fn=<TopkBackward0>)
top index: tensor([[0, 4, 1, 2, 5]])

i=0
for i in range(len(top_index[0])):
    print(f"Title candidate {i+1}: ", predicted_titles[top_index[0][i]], f' ---> probability = {float("{:.2f}".format(top_probabi

```

Title candidate 1: Nike | T-Shirt | Red ---> probability = 91.85%
Title candidate 2: Puma | T-Shirt | Red ---> probability = 6.98%
Title candidate 3: Nike | T-Shirt | Orange ---> probability = 0.59%
Title candidate 4: Nike | Shorts | Red ---> probability = 0.48%
Title candidate 5: Puma | T-Shirt | Orange ---> probability = 0.07%

Figure 5. 9: Top-5 title match with probability

5.3 Fine-tuning

Fine-tuning is required to improve the accuracy of attribute classification and title augmentation for each product. The process is targeted to achieve an end goal of accuracy greater than 95% for brand attribute and title augmentation. The CLIP model is considered as a baseline for fine-tuning. Out of several image encoders, the vision transformer (ViT) is selected for image processing and encoding. Based on the evaluation results in the previous section and datasets prepared for finetuning, we have decided to finetune the base model for two different paths: 1) finetune the base model with the brand dataset (auxiliary dataset) and create a checkpoint. Finetune further with the image-title dataset (primary dataset); and 2) finetune base model with the image-title dataset. These two paths are shown in Figure 5.10.

1) Finetuning with brand dataset and image-title dataset: For the first stage of finetuning, the brand dataset of image-text pairs is processed to feed into the training loop of the base model. The training is performed with a batch size of 200 for 10 epochs. The trained model is saved as a checkpoint after training is complete, and it is evaluated for brand accuracy and title generation accuracy. The saved model is further trained on the image-title training dataset. The process is similar to the previous step, it is trained for 10 epochs with batch size=98. The model is saved at this stage as another checkpoint after training is complete. An evaluation process is performed to check the improvement in accuracy for brand classification and title recommendation.

2) Finetuning base model with the image-title dataset: For this path of finetuning, only the image-title dataset is used. The training process is similar to the previous path, where the image-title dataset is processed to feed into the base model for training. The batch size is kept at 98 and training is performed for 10 epochs. The trained model is saved as a checkpoint and an evaluation process is performed to check the improvement in accuracy for brand classification and title recommendation.

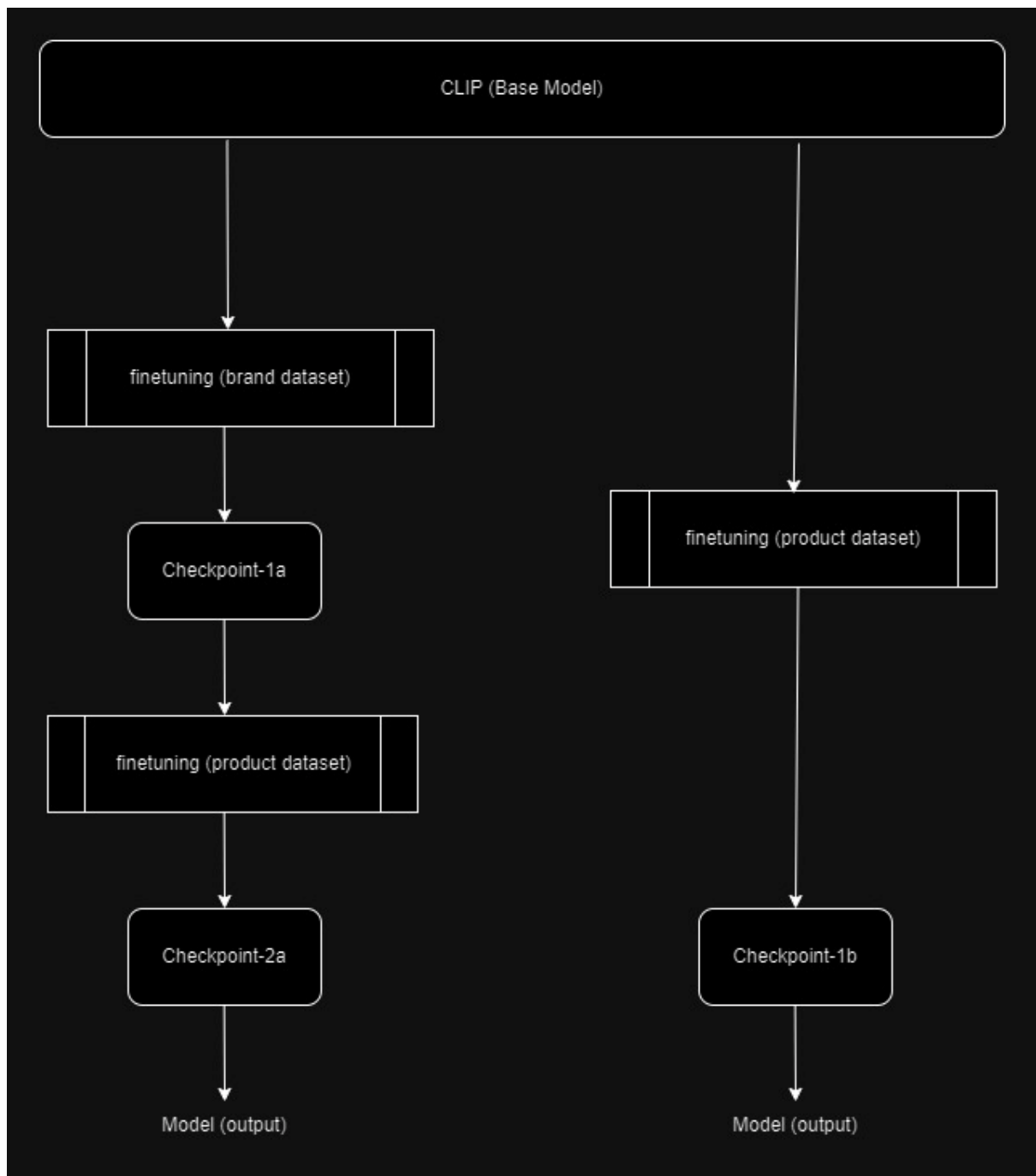


Figure 5. 10: Fine-tuning base model for two different paths

Chapter 6: Evaluation

The CLIP model is considered the base model and evaluated for zero-shot classification of brands, categories, and colors. The model is also evaluated for text augmentation of product images. These evaluation values are considered baseline for fine-tuning. The model is evaluated after each phase of fine-tuning and for different paths. The results are compared with the baseline to track the improvement in accuracy.

6.1 Evaluation of Base Model

The base model is evaluated for two variations of texts: 1) attributes without templates; and 2) attributes with templates. Attributes without templates contain the attribute class element as a single word. For example, if the category for a product is “Cap”, then the single word is considered for tokenization and embedding. Similarly, for other attributes single words are passed to the tokenizer and encoder. The same approach is followed when a product image is classified for text augmentation. The model gets an embedding value of shape (1, 512) for each class. The result for this variation is shown in Table 6.1. On the other hand, templates are used for the second variation, where we have a set of templates for an attribute as shown in Figure 5.2. There is a separate template for each attribute and custom-defined based on attribute values. The texts are formed by placing the attribute classes inside the templates. Each attribute class forms four sentences, which are passed to the tokenizer and encoder to form embedding values. The mean is calculated to form a standard embedding size of (1, 512). The evaluation result for this is shown in Table 6.2. The values for two variations are compared and it is concluded that the accuracy is better with templates. Based on this finding, we decided to use templates for all further classification and text augmentation operations.

	Top-1 accuracy	Top-2 accuracy	Top-3 accuracy	Top-4 accuracy	Top-5 accuracy
Brand Classification	30.00	42.86	50.95	55.24	59.05
Category Classification	90.48	99.52	99.52	100	-
Color Classification	83.81	93.33	96.67	98.57	99.05
Title Augmentation	32.38	41.43	45.24	48.10	49.52

Table 6. 1: Zero-shot attribute classification (without template)

	Top-1 accuracy	Top-2 accuracy	Top-3 accuracy	Top-4 accuracy	Top-5 accuracy
Brand Classification	36.19	45.24	51.90	58.10	62.38
Category Classification	99.05	100	-	-	-
Color Classification	83.33	95.24	97.62	98.57	99.52
Title Augmentation	35.24	44.76	49.52	53.33	55.71

Table 6. 2: Zero-shot attribute classification (with templates)

6.2 Evaluation of Checkpoint-1a

Checkpoint-1a is created after fine-tuning the base model with the brand logo dataset (auxiliary dataset). This is an attempt to increase the accuracy of brand attributes. The results are recorded in Table 6.3. This phase of fine-tuning results in reduced accuracy of brand classification and title augmentation.

	Top-1 accuracy	Top-2 accuracy	Top-3 accuracy	Top-4 accuracy	Top-5 accuracy
Brand Classification	25.71	33.81	40.0	42.86	44.76
Title Augmentation	4.29	7.14	9.05	10.48	10.95

Table 6. 3: Evaluation result of Checkpoint-1a

6.3 Evaluation of Checkpoint-2a

After fine-tuning with the brand logo dataset, checkpoint-1a is further fine-tuned with the product image-title dataset (primary dataset). This training phase increased the accuracy for both brand classification and title augmentation as shown in Table 6.4. The comparison with the baseline score is shown in Table 6.5.

	Top-1 accuracy	Top-2 accuracy	Top-3 accuracy	Top-4 accuracy	Top-5 accuracy
Brand Classification	32.38	45.71	52.86	58.10	64.29
Title Augmentation	36.67	47.62	52.38	57.14	58.57

Table 6. 4: Evaluation result of Checkpoint-2a

	Baseline	Checkpoint-2a	Improvement
Top-1 accuracy	35.24	36.67	+ 1.43 %
Top-2 accuracy	44.76	47.62	+ 2.86 %
Top-3 accuracy	49.52	52.38	+ 2.86 %
Top-4 accuracy	53.33	57.14	+ 3.81 %
Top-5 accuracy	55.71	58.57	+ 2.86 %

Table 6. 5: Comparison of Checkpoint-2a with the baseline

6.4 Evaluation of Checkpoint-1b

Checkpoint-1b is created after fine-tuning the base model with the product image-title dataset (primary dataset). The model is not trained with any other dataset. There are improvements in accuracy from the baseline for both brand classification and title augmentation. The results are shown in Table 6.6. The comparison of title augmentation accuracy with the baseline is shown in Table 6.7.

	Top-1 accuracy	Top-2 accuracy	Top-3 accuracy	Top-4 accuracy	Top-5 accuracy
Brand Classification	42.86	54.29	60.48	65.24	68.57
Title Augmentation	39.05	53.81	59.05	62.38	63.81

Table 6. 6: Evaluation result of Checkpoint-1b

	Baseline	Checkpoint-1b	Improvement
Top-1 accuracy	35.24	39.05	+ 3.81%
Top-2 accuracy	44.76	53.81	+ 9.05%
Top-3 accuracy	49.52	59.05	+ 9.53%
Top-4 accuracy	53.33	62.38	+ 9.05%
Top-5 accuracy	55.71	63.81	+ 8.10%

Table 6. 7: Comparison of Checkpoint-1b with the baseline

Chapter 7: Conclusion

This work focused on simplifying the process of text augmentation for product images by a fine-tuned multimodal model. The method can be used by small and medium-sized enterprises to their custom requirement by fine-tuning the pre-trained model for downstream tasks. Since the selected pre-trained model has zero-shot classification capability, it already performs benchmark accuracy for some attributes. The fine-tuning process is only focussed on the attributes which is not up to the mark, resulting in smaller and more relevant datasets.

The process of image and text similarity check is optimized by performing parallel comparisons of the image with each attribute, and then a batch of augmented titles is formed for final stage comparison before recommending a product title. This process reduces the size of each attribute list decided by a pre-determined hyper-parameter. The hyper-parameter value is changed to maintain an accuracy greater than or equal to 95% for each attribute. This approach optimizes the text augmentation process drastically which reduces the image comparison with a batch of 100,000 augmented titles to a batch of 20 augmented titles. So, it is highly recommended to tune the hyper-parameter for each attribute to control the batch size of the augmented text. It will keep the batch size under control even if the attribute list is expanded or new attributes are added to include in the product title.

The fine-tuning process is accomplished by primary and auxiliary datasets. The primary dataset contains image-title pairs and the auxiliary dataset contains brand logos with corresponding brand attributes. It is observed that the model accuracy is improved only with primary data and accuracy is diminished with auxiliary data. It is deduced that the image-title dataset is more appropriate for fine-tuning which gives the best result of Top-1 accuracy=39.05% and Top-5 accuracy=63.81%. This is a little improvement in the model accuracy by Top-1=3.81% and Top-5=8.10%.

Although the model shows some improvement from the baseline, it is far from the final goal of achieving an accuracy $\geq 95\%$. An increase in data size will increase the possibility of improving model accuracy, this may require collecting more product images from e-commerce platforms and defining product titles for them. More attributes can also be considered to form a bigger title for products. Additionally, extracted attributes can be consolidated in multiple ways to have a diverse set of augmented text.

References

- [1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781, 2013
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. arXiv:1706.03762, 2017
- [3] Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. arXiv:1405.4053, 2014
- [4] Keiron O'Shea, Ryan Nash. An Introduction to Convolutional Neural Networks. arXiv:1511.08458, 2015
- [5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929, 2020
- [6] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. arXiv:1908.02265, 2019
- [7] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. data2vec: A General Framework for Self-supervised Learning in Speech, Vision and Language. arXiv:2202.03555, 2022
- [8] Alec Radford, JongWook 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. arXiv:2103.00020, 2021
- [9] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. arXiv:2102.05918, 2021
- [10] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation. arXiv:2102.12092, 2021

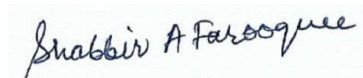
- [11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. arXiv:2002.05709, 2020
- [12] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A Survey on Deep Transfer Learning. arXiv:1808.01974, 2018
- [13] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources. arXiv:2306.09782, 2023
- [14] Ananya Kumar, Ruoqi Shen, Sébastien Bubeck, and Suriya Gunasekar. How to Fine-Tune Vision Models with SGD. arXiv:2211.09359, 2022
- [15] Abolfazl Farahani, Behrouz Pourshojae, Khaled Rasheed, and Hamid R. Arabnia. A Concise Review of Transfer Learning. arXiv:2104.02144, 2021
- [16] Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and Philip S. Yu. Multimodal Large Language Models: A Survey. arXiv:2311.13165, 2023
- [17] Ang Li, Allan Jabri, Armand Joulin, and Laurens van der Maaten. Learning Visual N-Grams from Web Data. 2017
- [18] Yuhao Zhang, Hang Jiang, Yasuhide Miura, Christopher Manning, and Curtis Langlotz. Contrastive Learning of Medical Visual Representations from Paired Images and Text. arXiv:2010.00747, 2020
- [19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners, 2019
- [20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165, 2020
- [21] Wonyoung Shin, Jonghun Park, Taekang Woo, Yongwoo Cho, Kwangjin Oh, and Hwanjun Song. e-CLIP: Large-Scale Vision-Language Representation Learning in E-commerce. arXiv:2207.00208, 2022

[22] Cem Akkus, Luyang Chu, Vladana Djakovic, Steffen Jauch-Walser, Philipp Koch, Giacomo Loss, Christopher Marquardt, Marco Moldovan, Nadja Sauter, Maximilian Schneider, Rickmer Schulte, Karol Urbanczyk, Jann Goschenhofer, Christian Heumann, Rasmus Hvingelby, Daniel Schalk, and Matthias Aßenmacher. Multimodal Deep Learning. arXiv:2301.04856, 2023

Declaration of Independence

I hereby certify that I have written this thesis independently and only using the specified sources and aids. The work has not been submitted to any other examination authority in the same or a similar form.

Berlin, 25.02.2024

A handwritten signature in black ink that reads "Shabbir A Farooquee". The signature is written in a cursive style and is placed on a light blue rectangular background.

Shabbir Ahmad Farooquee