Item Matching on the Shoppee Dataset - Final Report

Carlos Verastegui, Jinal Shah
George Mason University

December 9, 2021

Abstract

In today's world of global information at our fingertips, customers are flooded with virtually many options for product purchases. With so many options to choose products from, customers feel the best reward when they know that they have found the best deal for an item. With so many products with similar or distorted images and titles, it can sometimes be difficult to know if two items at different prices are actually just the same item. For this reason, e-commerce companies have taken the initiative to ensure customers that the item they're seeing is truly the cheapest amongst the sea of e-commerce. For this reason, Shopee, a popular east-Asian e-commerce site, released a Kaggle competition to see if machine learning methods can help aid in the problem of product matching. The goal is, given two samples, predict whether the two products belong to the same product group. Figure 1 belows shows example of matching products, as well as an example of label groups for products from the dataset. We propose utilizing deep learning techniques on two import features; image, and title. Our line of attack is to utilize these two features to create image and text embeddings, with the idea that we can utilize learned representations of these features in order to leverage traditional machine learning methods to determine similarity between products. In this paper, we outline the competition thesis, detail our approach to tackling the problem at hand, and then review our results. We perform ablation studies on various pre-trained architectures and training regimes to pinpoint successful models that perform well in product matching. Our results will show that while both image and text embedding models perform well in isolation, the concatenation of both surprisingly do not always yield better results. Additionally, we show other surprising results in the area of text embeddings, where we propose a different language model not used in the top scoring competition submissions, and show that it is able to consistently outperform other language models through a variety of ablations.

Keywords: Item Matching; Deep Learning; Kaggle Competition.

(a) Matching items example (b) output prediction example

Figure 1: Dataset examples

1 Introduction

Shoppers today have thousands of options to purchase products online. In such competitive markets it is important for these ecommerce companies to ensure correctness of product specifications and provide accurate product pricing. This would not only help in product categorization but also improve customer experience and retention. Thus, we have attempted to identify and classify such product postings on the Ecommerce platform Shopee, using the product images, titles and pHash of more than 34 thousand products using an unsupervised deep learning approach. We participated in the Kaggle competition sponsored by Shopee [**Shopee (2020)**]. Shopee is a multinational ecommerce technology company. In this project, we will be working on developing a model that will identify similar products that have been posted repeatedly on Shopee. It will also address similar grouping of the titles of those images that are uploaded on a daily basis to the site by resellers and individual dealers. Given the limited data for supervision in relation to the number of labels, we have come up with neural network models which boost the performance of the feature extraction of images and text. Moreover, we have also used transfer learning techniques to produce image and text embeddings to further improve the results.

Products uploaded by different sellers and individuals may not have a correct distinguishing feature such as UPC, GTIN, ISBN, etc, that can be used to identify similarity. Added to that there can be duplicate descriptions and many times it is not possible to find all the metadata of the product. The completeness of the product specifications and the taxonomies can vary across different platforms. In such situations, simply relying on identifiers and specifications can lead to an inaccurate identification of the product. Therefore, image matching using deep learning techniques can be promising. Not only, it can help set competitive pricing and enhance customer experience. There are many other use cases for an image matching model. It can also support accurate product categorization and uncover marketplace spam.

Figure 1(a) shows an example of the product that are the same but have been listed with different attributes. Using the specification of these products it would be impossible for an algorithm to find similarity between them. Therefore, image matching for such product listings can be very helpful.

1.1 Literature Review

There are few companies who have developed product matching models. EDITED is a market intelligence platform. They have used a data set of over 500 million product images to train the model using CNN algorithm on the preprocessed text and image vectors. Spotify's Annoy and Facebook's FAISS have also developed a model that uses Approximate Nearest Neighbors (ANN). Walmart uses techniques to identify existing products on their site versus products listed by the sellers [**More (2017)**]. They leverage cosine similarity metric on the dense layers of the network created from CNN and recommends using multiple models to test as they perform differently with the idiosyncrasies of the product image. Some of the techniques that they use are Concat CNN and Siamese LSTM.

There are several research papers and case studies that have analyzed different ML approaches [**Li et al. (2020)**]. For example, A study done in Yahoo labs also used deep convolutional Neural Network (CNN) to produce image embeddings. Another study focused on the efficiency of using BERT-based similarity that recommends employing BERT

models, using batch construction strategy. It emphasized on the concept Drift also known as covariate shift in the input and output variables during the process of transformation. Width.ai are the data science consultants widely use GPT-3 and SpaCy similarity models to find relationship between the sentence and the word placement. They have developed a custom image similarity model that is specific for the ecommerce retailers. The model is trained on variety of images that it is not necessary to include product images during training.

1.2 Dataset

We worked on the dataset that is provided by Shopee, which includes 34,251 images in JPEG format, spread out between 11,014 label groups. There is a csv file which includes details such as:

1. *posting_id*: The ID code for the posting
2. *image*: The image id/md5sum
3. *image_phash*: A perceptual hash of the image
4. *title*: The product description for the posting
5. *label_group*: The target label for all image postings that maps to the similar product. This will be used for training the model. Since this is the target label it is not provided in the test data set.

The Images and csv file will be used to train the network in the training phase. The trained model will then be used on the hidden test dataset of 70000 images on the Kaggle competition platform during an inference phase.

Input Data consists of Images, titles and pHash values. Output is the space delimited list of all posting Ids of the images that match other posting Ids of images. The grouping of the predicted images is capped at 50 similar images. In other words, similar images are grouped and maximum there can be 50 similar images in one group. Figure 1(b) shows an example of an output prediction.

1.3 Evaluation

For the evaluation of the model, we have used F1 score matrix. The given data set is imbalanced because of which accuracy matrix cannot be used to determine the efficiency of the model. It measures the number of correct predictions from all predictions made.

2 Details of the approach

To summarize, our approach consists of leveraging both the textual and visual information within each sample. On the visual side, we can create an encoding of the image that can represent the object within the image. On the textual side, we can create an encoding of the title that can represent semantic information about the object described. The hypothesis here is that in both problems, similar items will have a similar encoding, so we can leverage this encoding with traditional machine learning methods like k-NN to create a neighborhood

(a) Label group example (b) embedding example

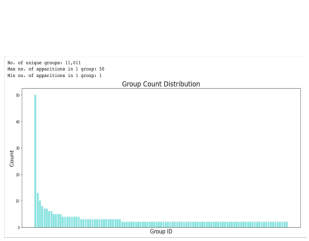
Figure 2: Approach examples

of similar items, and determine product grouping. Figure 2(a) shows an example of one such label grouping for a set of pajamas that describes the reasoning for our approach. All six images contain the pajamas in some aspect. Whether folded, rotated, different coloring, etc, all six images have the product in a clear visual manner. Additionally, the title for each pajama item shown in the images, all contain similar words that describe the object. It is through this method encoding that we hope we can identify all these products as the same.

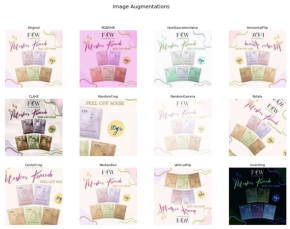
We describe the general encoding method we used as a form of embedding [**Koehrsen (2018)**]. Embeddings are a common approach to encoding objects in machine learning, as it makes it easier to understand abstract objects through a mathematical lens. Embeddings are simply a learned representation of an object, such as text or images, in a real-valued vector space. What this means is that an object is mapped by some model f to a vector in a vector space. The goal here is that this vector encodes information about the object, so that all the necessary semantics about our object are represented mathematically. Figure 2(b) shows an example of two sentences which are mapped to a hypothetical vector space. We see here that the two sentences are similar in a semantic context, and as such the words that have similar meaning are mapped to similar points in the vector space. This is the crux of our hypothesis; it is through these learned embeddings that we hope in the vector space similar objects will be mapped to similar points. As such, this allows for a spatial machine learning like k-NN to be useful as to predict objects in the same label group. For text, we design a model that learns how to represent text as a vector, and likewise for images we design a model that learns how to represent images as a vector. We describe our approach with embeddings in detail by specifying our pre-processing, as well as the individual approaches for image and text.

2.1 Pre-processing

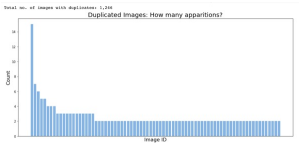
The pre-processing of the dataset meant understanding the distribution of the label groups along with text and image discrepancies [**Olteanu (2020)**]. The first thing to notice is how the label groups are distributed. It is to be expected that there can be a difference in the amount of products between label groups, which Figure 3(a) shows the distribution. We see here that lower product group IDs have a far larger amount of products within groups, up to about 50 products per label group. The majority of label groups, however, contain less than 5 images per label group. This contributes to the outcome of our approach, as label groups with few products will have a smaller amount of samples to train from. Having the label group distribution contain outlier groups, we felt it be necessary to balance the



(a) Label group distribution



(b) Data augmentation examples



(c) Duplicate image distribution

Figure 3: Pre-processing results

dataset by providing data augmentation via imaging [**PyTorch (2020)**]. Figure 3(b) shows an example of data augmentation on one product, where we append to our dataset copies where the title is the same, but the image is now flipped, saturated, blurred, cropped, and so on. This helps mitigate the difference in densities between label groups, by making the amount per label group larger, so that the effects of the anomaly label groups are mitigated. In addition to augmentation we have normalized the input images. This would ensure that the pixels of images will have similar data distribution so that during back propagation the weights converge faster. In this technique we subtract mean from each pixel value and divide it by the standard deviation.

Ideally, each label group given in the training dataset would identify its corresponding similar images. However, there are many duplicate images. With the close investigation it was found that some images had same titles but were in different label group and some images had same label group but had a very different title. Figure 3(c) represents number of duplicate images, which were removed before model training.

2.2 Image Embeddings

We leverage the utility of image embeddings as part of our approach [**Beaumont (2020)**]. There are multiple different models tried on the preprocessed images. Initiated with the CNN as the base and implemented our model by adding more layers to it. In an image are thousands of features that needs to be evaluated. These key points or features is what distinguishes the object within it. Key points are broken down and its weights are calculated based on the pixel density. Every small feature is broken down into multiple filters which combine and form the one feature. The image is broken into multiple filters, for our custom model. We have used 64 filters for each image. Along with the filters, we set the grid or the kernel size of 3 x 3 pixel which multiples with the original image pixels and creates a feature map. When the number is closer to 1 in the feature map means there is a feature present on that pixel that gets activated. So, in short multiple filters are aggregated together to help create a feature map and they detect the features out of an image using the set grid size. Convolution is a linear operation that multiplies the set of input weights of an image, or dot product between the input array of pixel values of a patch on the image and its weights (filters). This dot product is summed and results in a single value also known as "scalar product". The filters are usually used in an overlapping manner over a set of pixel patches on the input image so that it covers an entire image from left to right and top to bottom. This allows to capture the feature that can be present anywhere in the image.

To reduce feature maps and dimensions Adaptive average pooling was advantageous, since we did not have to specify the stride and kernel size as hyperparameters. Instead we just specify the output size and other hyperparameters are automatically adjusted. In the final step we flatten the different feature maps into a one-dimension array. Figure 9 shows the architecture for our custom model.

In addition to trying out a custom model, we also leverage transfer learning for better results. Transfer learning is a technique in machine learning where we can take a pretrained model developed on one task and use it as a starting point in some other similar but different tasks. It saves lot of computational power and training time. We tried using pretrained models from Pytorch and customize it for the given use case. Some of the models we customized and tried using on the images includes MobileNet, ResNet50, NFNet and an Ensemble of ResNet + EfficientNet+ NFNet. Below we give brief overviews of the pre-trained models used.

ResNet [**He et al. (2015)**]is an architecture that attempts to avoid vanishing gradient problems by introducing skip connections. Skip connections in these networks allow us to take the activation from one layer and feed it to another layer even much deeper in a neural network. This has shown to improve our score score by almost 5% F1. EfficientNet [**Tan and Le (2019)**] systematically scales the depth, width and image resolution of a CNN to balance the dimensions of width, depth and resolution by scaling with a constant ratio 2n and using alpha, beta and gamma values. This is similar to NAS in the sense that EfficientNet attempts to search for an efficient CNN model architecture. We have used Efficient Net B0 and B2 classes and did not include the fully-connected layer at the top of the network instead specified the input shape and pooling. We also added our own pooling and dense layers. Finally, NFNet [**Brock et al. (2021)**] is a high-performance neural network without normalization. Compared to other networks there is no batch normalization used. Batch Normalization is an important technique in training the neural network, however there are some disadvantages to this technique which are superseded in this model, such as demands for large batch size for higher accuracy and discrepancy between training and inference. To overcome this NFNet proposes using scaled weight standardization to compute the mean and standard deviation of the weights instead of the data, as well as adaptive gradient clipping to make intelligent steps of the gradient. The underlying architecture of NFNet is ResNeXt, which can be considered an extension of InceptionNet utilizing skip connections. Other architectures were used, such as MobileNet and InceptionV3, but they performed significantly worse than our custom baselines model, so we excluded them. Figure 8 shows the pipeline used for pre-trained architectures.

In addition to the images, the pHash values are given in the training and test dataset so using it during the testing does help in the results by 0.1%. Perceptual hashing is an algorithmic approach to produce a snippet or fingerprint of an image. In other words, Perceptual hash values are unique to each image based on its features. So, these values will be identical and comparable if features of the image are similar. Perceptual hash values represent the image's visual appearance and do not get affected by rotation, skewing and contrast adjustments. Thus, they are invariant to certain feature transformations and are highly reliable for image classification. These values are grouped and concatenated during the prediction stage along with the image and text embeddings. Finally, we combined ResNet50, EfficientNetB0 and NFNetL0 pretrained models. Then we compute final image embedded values by taking the average of all the features extracted from these models.

These image embeddings will be further used to find the similarity between the multiple images during an inference stage.

2.3 Text Embeddings

Each sample in the dataset contains a *title* feature, which is of course the title of the product on the Shopee website. As such, the title of the products can be used to perform product matching under the hypothesis that identical products will contain similar product titles. Our approach as outlined above is to first devise an architecture that takes text as input and outputs an embedding of the text. The next phase is to take this matrix of text embeddings for each title and fit the dataset into a nearest neighbors model using cosine similarity, where finally we consider two titles to be in the same label group if their cosine similarity is above a certain threshold. In the spirit of the objective of the paper, we wish to utilize a deep learning approach for our architecture, and perform ablation studies on the various approaches to designing an embedding model.

A natural choice for a textual embedding model is utilizing a language model. A language model can be thought of as the first stage of a transformer, where we first take text and then output a text embedding. Such language models include BERT, RoBERTa, DistillBERT, and so on. What makes language models powerful tools is that, much like the name suggests, it creates a semantic framework from the training data. It is here that we can leverage the power of deep learning and transformers to design a model where all the text in the dataset can be holistically viewed such that the language model can learn relationships between words not just sequentially, but within the bigger context of the sentence itself. It is for this reason that language models have been so successful in areas of NLP beyond their traditional problems such as translation and text generation. Language models in this context are used for semantic text similarity, where we wish to determine semantic similarity between text. Our architecture is simple then, as we use a language model as our backbone to design an embedding, which is then passed through a max pooling layer and then finally a linear layer to output a text embedding of 768 dimensions.

We perform an ablation study on the architecture by mainly surveying different language models, and plotting their performance against each other. Other ablation studies are performed on the top performing models to fine tune their performance and make their distinctions clearer. In the interest of time, we tested out four different state-of-the-art pretrained language models, all retrieved from Huggingface: DistilBERT-base-indonesian [**DistilBERT (2019)**], DistilRoBERTa-base [**DistilroBERTa (2019)**], XLM-multilingual-r-v1 [**XLM (2019)**], and MPnet [**MPNet (2020)**]. We note here that DistilBERT-base-indonesian has been the frequent top-most performer of all the top notebooks, so we compare all other language models with respect to this one. It is important to note here the contrast of the textual task from the image task: here, different languages can be considered. Indonesian was found to be successful due to Shopee being an east-Asian e-commerce site, where not that many titles may be in English.

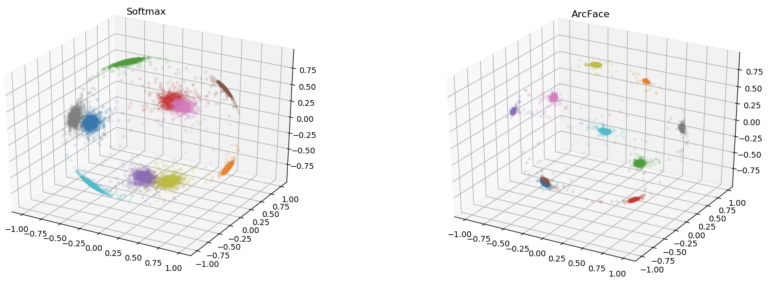
To summarize the different language models, we start of by describing BERT [**Devlin et al. (2018)**], which is the original language model which is a bi-directional transformer that is pre-trained on the tasks of Masked Language Model and Next Structure Prediction. This is done by masking certain tokens from the dataset to predict the word that might appear. RoBERTa [**Liu et al. (2019)**] is a variant of BERT, where Next Structure Prediction is removed from the pre-training tasks and dynamic masking is introduced, where

the masked tokens differ throughout each training epoch, which found itself performing better than traditional BERT. DistilBERT [**Sanh et al. (2019)**] is another variant of BERT, that learns an approximate solution as BERT through a technique called distillation that approximates the neural network of BERT, using far fewer parameters and no token embeddings. This results in DistilBERT being much faster than BERT, and obtaining very similar performance to BERT. Consequently, DistilroBERTa is a distilled version of RoBERTa, trained in the same fashion. XLM [**Lample and Conneau (2019)**] utilizes BERT along with a pre-processing technique called Pair-Byte encoding to train on multiple languages. This is done so that XLM can learn relations between words in different languages, which can perform far better than other language models like BERT on multi-lingual tasks. Finally, MPnet [**Song et al. (2020)**] combines the permuted language model pre-training task of another language model called XLNet, which is an improvement upon the masked language modeling of BERT, with auxiliary position information of the tokens to take the full training sentence in context. All of these language models, with the exception of XLM-multilingual, were chosen due to their overall success in language modelling tasks and their frequent appearance in top scoring submissions. We propose XLM-multilingual into this ablation as our own finding despite it not being a common choice among other competitors, and show that this language model is able to outperform all other three, which we attribute due to leveraging its designed task of performing well on multilingual data. One thing we will point out is XLM-multilingual takes more time to fine-tune.

For our specifics on how we trained the textual model, we utilized ArcFace [**Deotte (2020)**] as our loss, and used cosine annealing with a warm-up of two epochs [**Kim (2020)**]. The model was trained for up to 10 epochs as the performance began to plateau by the 5th epoch. ArcFace as a loss was used over other common losses such as cross-entropy because ArcFace was more appropriate for the task at hand. ArcFace leverages the fact that if we are to normalize every embedding, what we get a unit-sphere in N-dimensional space, where N is the length of our embedding. It is here where ArcFace can add to our loss by encouraging similar classes to be closer in the unit-sphere, and dissimilar classes to be farther away. Figure 4 shows an example of our normalized embeddings without ArcFace (left), and with ArcFace (right). Additionally, we utilize Sharpness-Aware Minimization [**Foret et al. (2020)**] as a wrapper for our optimizer as part of our ablation. We note that SAM is used frequently in many SotA models for image and text embeddings, including some top scoring competition submissions, as it simultaneously minimizes loss and loss sharpness by seeking parameters with uniformly low loss, which can lead to wider minimums and thus better generalization. For this reason we felt it would be necessary to include it in our ablation studies.

3 Results

Similar to describing our approach, we divide our results into that of our image embedding approach, as well as our textual approach. We conclude this section with highlighting our performance when we concatenate the best performing model from each section into one pipelined architecture, and see how it compares to isolated approaches. The code to our complete model is contained in one single Python Notebook ran on Kaggle. Both the image and text architectures and the kNN for each are within the notebook as to easily combine



(a) Unit sphere without ArcFace (b) Unit Sphere with ArcFace

Figure 4: Difference in applying ArcFace loss

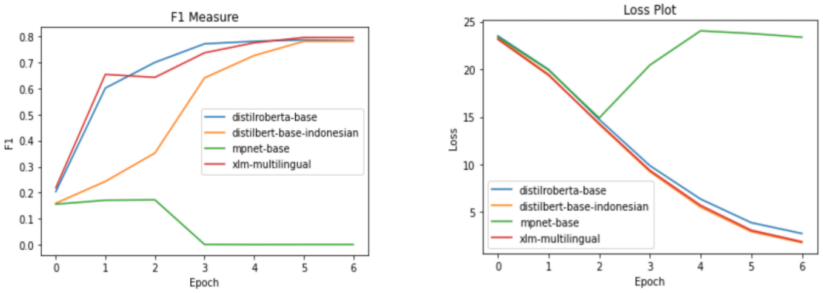
the results and submit to Kaggle. To avoid submission limits, we submitted to Kaggle the image models along with the concatenated models to get results on the private testing data. The results explained in the Text Results section describe results on the training data split into training and validation sets. The dataset used of course is the Shopee Item Matching dataset provided by the Kaggle competition.

3.1 Image Results

Figure **??**(c) shows the results of various ablation studies submitted to Kaggle with their score. From various experiments that were performed on this dataset, it was found that ensemble techniques, NFNet, and NFNet combined with XLM-Multilingual text classification model outperformed other approaches by giving us a top F1 score of 0.729. The improvement in the score using these specific techniques can be because of the scaled weight standardization mechanism of the NFNet, that computes the mean and standard deviation of the weights itself instead of computing it from the input features. Batch normalization in the ResNet architecture required to have set large batch sizes of input images during training. This could have negatively affected the training especially because some of the images had text overlays and many variations that required detailed feature extraction that are possible with small batch sizes. On the other hand, the scaling effect of the EfficientNet did help improve the score in comparison to the custom model and ResNet, however when combined with the NFNet in the ensemble model training, the score did not improve. Adding Angular Margin Loss in the last layer helped improve the score by 0.02. ArcFace loss is widely used in face recognition tasks. In this specific use case it worked well because there are many images that have facial features. One distinguishing feature of the NFNet that contributed to increasing the score drastically is its ability to clip gradients. This particularly helps with finding global minima with reduced loss and better predictions.

3.2 Text Results

For text results, we describe our results from trying the various language model backbones, and then the results from performing further ablation studies on the best scoring language model backbones. Figure 5 contains the F1 measure and loss for each model across all 6 epochs. We can immediately see that an outlier of this group is the MPNet-base model, which performed extremely poorly at near 0.0 F1 and a loss of 25. Once the warmup



(a) F1 Measure Plot (b) Loss Plot

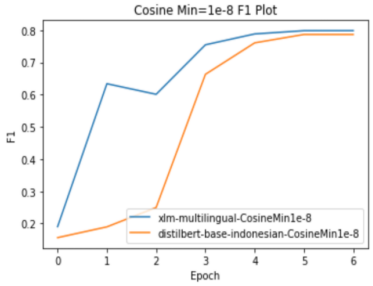
Figure 5: F1 and Loss for all four language model backbones

epochs end it diminishes in performance, which we postulate at the fact that the MPNet-base pre-trained backbone is solely trained on an English dataset, which combined with a variable cosine annealing loss would make the model behave very poorly. Additionally, perhaps pre-training tasks such as permuted language modelling may be the reason the models behaves poorly on the dataset. We note that even changing loss minimums and maximums, or removing SAM did not improve the model, so we consider it an anomalous case.

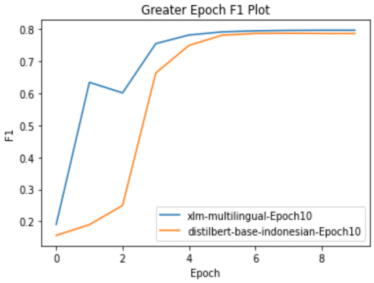
Otherwise, the other three backbones performed at similar rates, with all nearing a 0.8 F1 measure on the validation set. Both classes of distilX backbones had the advantage of training much quicker, due to the distillation methods giving approximate predictions of an actual neural network used in a more complete language model. Nevertheless, the two performed very well, with surprising results from distilroBERTa being that it was pre-trained on an English dataset. However, we do note that distilroBERTa performed slightly worse than xlm-multilingual and distilbert-base-indonesion in terms of loss, coming in at about a unit worse. Xlm-multilingual, taking much longer to train than the distilX backbones, was able to perform very well, most likely due to it leveraging cross-lingual pre-training to account for the Indonesian and English text within the dataset. It performed on par with the SotA distilbert-base-indonesian, so we isolated the two for further ablations and compared the results.

Figure 6 shows the F1 measure results from performing further ablation studies on distilbert-base-indonesian and xlm-multilingual. We can see that across all four ablations, xlm-multilingual was able to out-perform the SotA distilbert-base-indonesian language model backbone. We suggest that our finding of utilizing xlm-multilingual is able to improve the scores of the top performing competition submissions if swapped for that language model, instead of using distilbert-base-indonesian. We explain each ablation in detail.

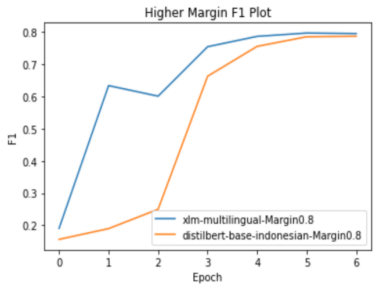
Top right of Figure 6, we lowered the learning rate minimum of the cosine annealing from $1e-5$ down to $1e-8$, as we noticed that smaller learning rate values lead to slightly better performance across all models, improving their scores by about 0.5% F1. Xlm-multilingual was able to clearly outperform distilbert-base-indonesian by about 0.3% F1. We kept the learning rate at the new minimum and performed the rest of the ablations. Top left we extended the number of epochs from 6 to 10 to see if longer training led to some jumps in performance, which we see that at epoch 6 the F1 begins to plateau, making no difference for future epochs. Bottom left we increased the angular margin penalty on the



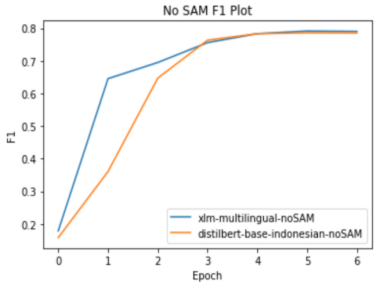
(a) Lowered Cosine Minimum F1



(b) Longer Epoch Training F1



(c) Higher ArcFace Margin F1



(d) No SAM F1

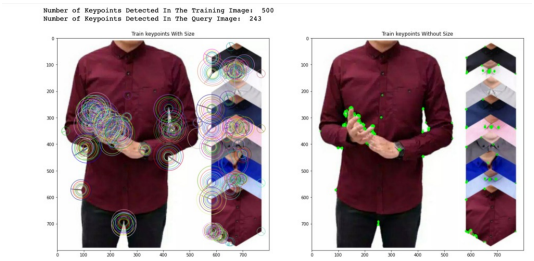
Figure 6: F1 Measure plots for various ablations

target angle for ArcFace loss from 0.3 to 0.8, after testing higher values and found 0.8 to perform objectively the best across all models. Finally, bottom right we remove the SAM optimizer wrapper and trained both models on just the Adam optimizer. It is clear that xlm-multilingual is able to fully utilize its cross-lingual pre-training to take into account the different languages in our dataset, thus outperforming the SotA language model. As such, we have kept xlm-multilingual as part of our final model for text embeddings, and for concatenation with the image embeddings.

3.3 Concatenated Results

Once we obtained the best performing models from both problems, we created an ensemble model that takes the union of both label group findings as its final prediction. This is in contrast to other contest submissions that instead concatenate the embeddings from each problem, and then run kNN on the embeddings to produce label group predictions. We found from surveying other submission results that the order in which the neighboring was performed did not matter, as both performed the same. We then felt is was appropriate to stick with the former since it would utilize the best performance of both models in order to concatenate both findings. Figure 11 shows the results of models 6 and 9 that combine the top two best scoring image embedding models with the best scoring text embedding model. We see that combining ResNet50 and xlm-multilingual brought our private Kaggle score from 0.64% F1 up to 0.68%, a near 4% improvement. However, combining NFNet with xlm-multilingual caused the model to perform 0.09% worse than NFNet alone. We consider the reason for this to perhaps be that because NFNet is able to achieve 0.72% F1, the additional label groups from xlm-multilingual could point to false positives, possibly due to overfitting the language model to the training set. Additionally, we provide a contrast of

(a) Matching points between two images in the same label group



(b) Keypoint detection on an image

Figure 7: Examples of ORB performed on the training data

an ensemble model consisting of the three best image embedding models, and found it to perform worse than NFNet. Here, we consider the fact that NFNet has nearly a 0.1% F1 improvement from ResNet and EfficientNet, so perhaps the extra voting of the two lower performing CNN models would saturate the performance of NFNet altogether.

Overall, we conclude that our concatenated model generally improves over isolated image or text embeddign models, which brings our private F1 score to 0.72%, on par with other lead submissions. This puts our NFNet image model in the top 100 submissions out of all 2300 submissions, and our concatenated model in the top 500 of all 2300 submissions.

4 Discussion and Conclusions

In conclusion, our results from our ablation studies show general improvements across other leading submissions for the competition. Inspecting the top scoring submissions show more advanced machine learning techniques, such as alpha-Query expansion, distributed model embeddings, and agglomerative clustering. Our modest ensemble model was able to offer competitive results, due to ablation studies on various backbones and changes to our training regime. We believe that our negative results could've came from overfitting of our training data, especially in the context of language modeling where the model may be too exposed to just the language that it sees in the training set that is not indicative of the entire language itself.

Further ablation studies can be performed in the future related to concatenation techniques in order to improve the model. Such techniques can include combining embeddings and furthering the pipeline for more sophisticated techniques like keeping concatenated and isolated embeddings separate, or using dynamic clustering techniques like DBSCAN to find neighborhoods of label groups. We propose in this context it would be difficult to use traditional clustering techniques such as K-means as we are not certain of the possible label groups, thus any arbitrary choice of K could lead to false results. Additionally, noting overfitting of language models could lead to a possible solution being to have multiple language models from a diverse set of language groups be ensembled to give out different embeddings, and then be concatenated. We postulate that this could reduce the problem of overfitting of language modeling by offering different semantic perspectives, than using just one language model. Additionally, future solutions could extend beyond just deep learning, and perhaps use traditional computer vision methods such as SIFT, ORB, and SURF to

define similarities between images. Such algorithms can leverage the fact that they are designed to detect local features in images, and as such are able to possibly perform very well in detecting similar features in images that may indicate identical products. This could yield competitive results to the embedding approach that we have attempted, and can even be ensembled with our approach as offering a different perspective of local searching instead of global embedding. Figure 7 shows examples of utilizing ORB on the training data, which we can see does a sufficient job at detecting local features.

Overall, we find that our ablation studies have been successful in identifying possible improvements to both the text and image embeddings frequently tried on the competition. Our results have gained us valuable insight into the problem of product matching and finding similarities between objects by highlighting key issues in the approach of global embedding, and we hope that our findings can help guide future contestants in streamlining choices of CNN and language models to use as their backbone, along with useful training improvements such as ArcFace parameters and SAM for optimizer wrapping.

5 Statement of individual contribution

This project was done by Jinal Shah and Carlos Verastegui. The work was evenly split between between the two, with Jinal taking over the image embeddings and Carlos taking over the text embeddings. Both members wrote their appropriate sections of this paper, with the introduction being written by Jinal, and the abstract and conclusions written by Carlos. All other sections of the paper were a combined synthesis of what people members wrote for that section.

In addition, we briefly outline additional code that we used throughout the project that was not mentioned in the paper here. In general, we surveyed various successful Kaggle notebooks such as [**mfalfafa (2020)**], [**kurupical (2020)**], and [**smartdolphin (2020)**] to see how their approaches compared, and used some of their code to test out our own ablations. Additionally, we used some code described in [**Olteanu (2020)**] to pre-process our data and visualize it. Pre-trained CNN architectures were fetched form the PyTorch library, and language models were fetched from Huggingface, as described above. We utilized the code found at [**Mavani (2020)**] for creating the ArcFace class, and [**Kim (2020)**] For the learning rate warmup with cosine annealing. Some code in our notebook is universal, such as dividing the embeddings into chunks to get the similarity or getting the label groups as part of the pre-processing and F1 scoring, as the code recipe for it was found in every notebook we looked at. Our notebooks are included with this paper.

References

Beaumont, R. (2020). Image embeddings. https://rom1504.medium.com/image-embeddings-ed1b194d113e.

Brock, A., S. De, S. L. Smith, and K. Simonyan (2021). High-performance large-scale image recognition without normalization. *CoRR abs/2102.06171*.

Deotte, C. (2020). Embeddings, cosine distance, and arcface explained. https://www.kaggle.com/c/shopee-product-matching/discussion/226279.

Devlin, J., M. Chang, K. Lee, and K. Toutanova (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR abs/1810.04805*.

DistilBERT (2019). cahya/distilbert-base-indonesian. https://huggingface.co/cahya/distilbert-base-indonesian.

DistilroBERTa (2019). distilroberta-base. https://huggingface.co/distilroberta-base.

Foret, P., A. Kleiner, H. Mobahi, and B. Neyshabur (2020). Sharpness-aware minimization for efficiently improving generalization. https://github.com/davda54/sam.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Deep residual learning for image recognition. *CoRR abs/1512.03385*.

Kim, I. (2020). pytorch-gradual-warmup-lr. https://github.com/ildoonet/pytorch-gradual-warmup-lr.

Koehrsen, W. (2018, 10). Neural network embeddings explained. https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526.

kurupical (2020). 6th place solution. https://www.kaggle.com/c/shopee-product-matching/discussion/238010.

Lample, G. and A. Conneau (2019). Cross-lingual language model pretraining. *CoRR abs/1901.07291*.

Li, F., S. Kant, S. Araki, S. Bangera, and S. S. Shukla (2020). Neural networks for fashion image classification and visual search. *CoRR abs/2005.08170*.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR abs/1907.11692*.

Mavani, V. (2020). Shopee competition: Arcface. https://www.kaggle.com/vatsalmavani/eff-b4-tfidf-0-728.

mfalfafa (2020). Silver medal solution for shopee. https://github.com/mfalfafa/shopee-price-match-guarantee.

More, A. (2017). Product matching in ecommerce using deep learning. https://medium.com/walmartglobaltech/product-matching-in-ecommerce-4f19b6aebaca.

MPNet (2020). sentence-transformers/all-mpnet-base-v2. https://huggingface.co/sentence-transformers/all-mpnet-base-v2.

Olteanu, A. (2020). Shopee competition: Eda and preprocessing. https://wandb.ai/andrada/shopee-kaggle/reports/I-Shopee-Competition-EDA-and-Preprocessing--Vmlldzo1NDEzNzU.

PyTorch (2020). Pytorch image augmentation using transforms. https://androidkt.com/pytorch-image-augmentation-using-transforms/.

Sanh, V., L. Debut, J. Chaumond, and T. Wolf (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR abs/1910.01108*.

Shopee (2020). Shopee - price match guarantee. https://www.kaggle.com/c/shopee-product-matching.

smartdolphin (2020). Shopee - price match guarantee. https://github.com/smartdolphin/shopee.

Song, K., X. Tan, T. Qin, J. Lu, and T. Liu (2020). Mpnet: Masked and permuted pre-training for language understanding. *CoRR abs/2004.09297*.

Tan, M. and Q. V. Le (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR abs/1905.11946*.

XLM (2019). sentence-transformers/paraphrase-xlm-r-multilingual-v1. https://huggingface.co/sentence-transformers/paraphrase-xlm-r-multilingual-v1.

6 Appendix

We include here additional figures here for clearer visualizations.

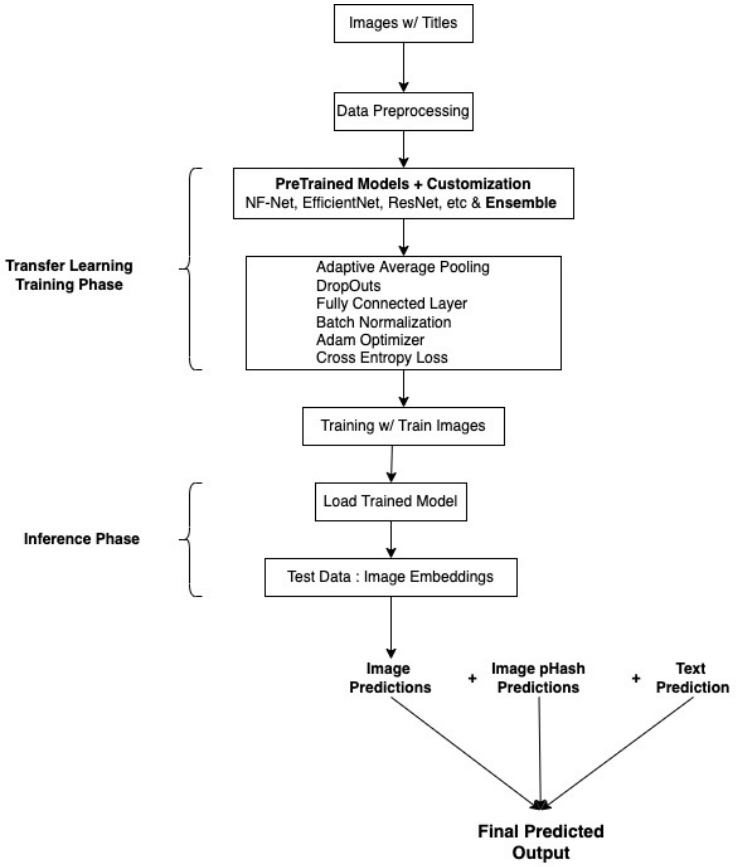


Figure 8: General pipeline for image and concatenation embeddings

Custom Model for Matching Products								
	Residual Block	Layer Type	In Channel | Out Channel	Feature Maps	Size	Kernel Size	Stride	Padding
Feature Extraction	1	Convolution	3 | 64	64	227 x 227 x 3	3 x 3	1	1
		AdaptiveAvgPool	64 | 64	64	114 x 114 x 64	2 x 2	2	1
		Batch Normalization			114 x 114 x 64			
		ReLU			114 x 114 x 64			
	2	Convolution	64 | 128	128	114 x 114 x 128	3 x 3	1	1
		AdaptiveAvgPool	128 | 128	128	58 x 58 x 128	2 x 2	2	1
		Batch Normalization	128		58 x 58 x 128			
		ReLU	128		58 x 58 x 128			
	3	Convolution	128 | 256	256	58 x 58 x 256	3 x 3	1	1
		Convolution	128 | 256	256	58 x 58 x 256	3 x 3	1	1
		AdaptiveAvgPool	256 | 256	256	30 x 30 x 256	2 x 2	2	1
		Batch Normalization	256		30 x 30 x 256			
		ReLU	256		30 x 30 x 256			
	4	Convolution	256 | 256	256	30 x 30 x 256	3 x 3	1	1
		Convolution	256 | 256	256	30 x 30 x 256	3 x 3	1	1
		AdaptiveAvgPool	256 | 256	256	16 x 16 x 256	2 x 2	2	1
		Batch Normalization	256		16 x 16 x 256			
		ReLU	256		16 x 16 x 256			
	5	Convolution	256 | 512	512	16 x 16 x 256	3 x 3	1	1
		Convolution	512 | 512	512	16 x 16 x 512	3 x 3	1	1
		AdaptiveAvgPool	512 | 512	512	9 x 9 x 512	2 x 2	1	1
		Batch Normalization	512		9 x 9 x 512			
		ReLU	512		9 x 9 x 512			
		FC	num_epochs : 20 batch_size : 32 learning Rate: 0.000024 Number Of Classes : 11014		51200			
		ReLU			51200			
		DropOut			na			
		FC			1024			
		ReLU			1024			
		DropOut			na			
		Output - FC			11014			

Figure 9: Custom model architecture

Product Matching			
#	Model	Feature / Variations	F1 Private Score on Kaggle
1	CNN	Epoch : 25 Layers : 3 Activation Function : ReLU Loss : Cross Entropy Optimizer : Adam	0.54
2	Mobile Net	Epoch : 25 Optmizer : SGD	0.56
3	InceptionV3	Epoch : 25 Optmizer : SGD	0.52
4	Custom Model	num_epochs : 20 batch_size : 32 learning Rate: 0.000024 Loss : Cross Entropy Adam Optimizer	0.616
5	ResNet50	Same Parameters	0.647
6	ResNet50 + XLM-MultiLingual		0.68
7	EfficientNet_b0	Same Parameters	0.647
8	NF-Net	Same Parameters	0.729
9	NF-Net + XLM-MultiLingual	Same Parameters	0.72
10	Ensemble : ResNet + EfficientNet + NF-Net	Same Parameters	0.713

Figure 10: Ablation results for image embeddings

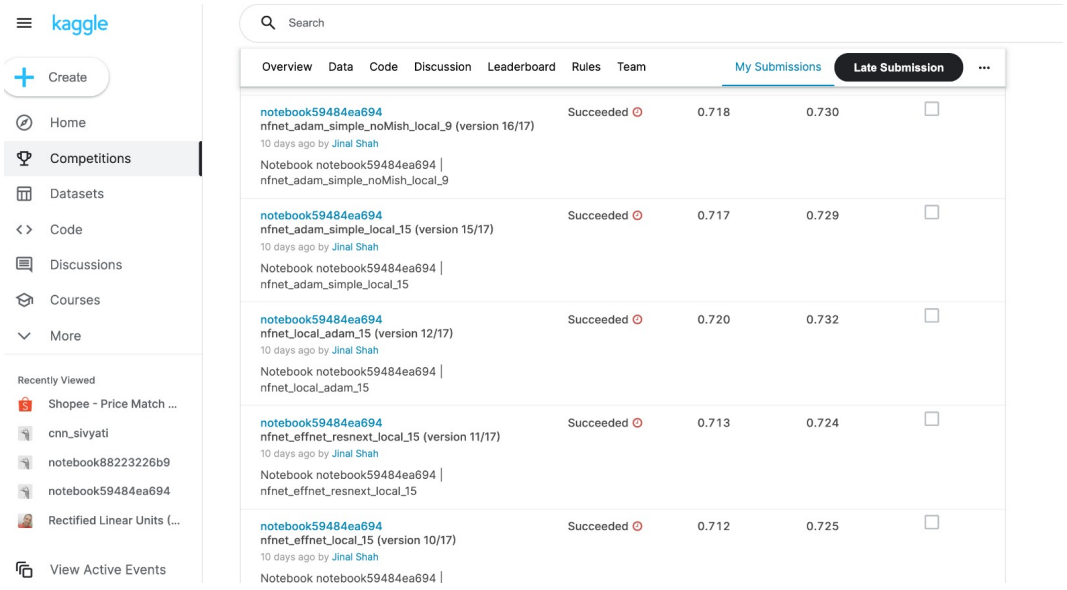


Figure 11: Results from submitted Kaggle notebooks