Project Report : CS 7643 Spring 2021
Multi-modal Representation Learning for Product Matching

Ashwani Gupta, Chungjin Lee, Jungho Kim
Georgia Institute of Technology
ashwani@gatech.edu, chungjin.lee@gatech.edu, jkim3302@gatech.edu

Abstract

In this paper, we present a novel approach to do product matching task, multi-modal representation learning. As product matching is usually done using either image or text alone, our motivation is to make use of both information. While using pre-trained CNN and transformer model for feature extraction, we used Siamese network structure to learn task-specific representation. For training, we used binary classification task to given product posting pairs and used k-NN algorithm to do product matching task given learned representation. This is the concept of surrogate task such that we trained the Siamese network to learn better representation, starting from embedding of pre-trained models. Lastly, by using concatenation of text and image embedding, we were able to represent product information in task-specific manner, grouping similar items together. As product matching (grouping) is essential task for e-Commerce industry, we expect the result of this paper will contribute the industry and other research on similarity matching.

1. Introduction/Background/Motivation

A perfect product matching is the key to success for any e-commerce company, because end-users want to have the lowest price and best quality. When searching for a product, a user would like to see other matching options. A fast and accurate search of quality product with low prices would attract customers to e-commerce portals with such functionality. Therefore, a company offering a perfect product matching gives an edge over other competitors. If we are successful, it would give a huge ease to the customers. In this paper, we have introduced a novel model which combines image and text representation using Siamese network.

Having Joined a Kaggle competition hosted by a Southeast Asia based retail company, Shopee, we believe this project is directly originated from the industry's needs. For various e-commerce domains, this project has huge potential contributions. The most straightforward use would be product grouping which attempts to automatically group items based on their attributes, images and texts in postings. This seems to be simple but might be delivered into many services from including price comparison, related product recommendation, redundant postings grouping. Furthermore, if our approach gives successful results, the model can represent each item as an embedding vector which, for other related researches, can be actively used as items' vector representations.

1.1. Task

A product-matching machine learning model would encounter completely unseen images and text descriptions. Major difficulty for such automation is that many unseen items are being posted by individual sellers in real time[11]. Therefore, the task cannot be solved using supervised methods, which relies on human annotation which demands lots of monetary and human resources. This competition is code competition such that competitors, including us, can upload their code and only uploaded code can access and make prediction to test data.

Lastly, this problem can be categorized as Zero-shot Learning, in which for testing a model observes a set of novel classes which haven't been provided to the model during training. Although in practical product matching, some of postings would have labels from training set, the competition's test set doesn't have classes from training at all.

1.2. Data Exploration

Given 34K training data which consists of both product image and title information, we would like to train a model and eventually cluster $posting\_id$ to $label_group$1.

posting_id	image	image_phash	title	label_group
ID for posting	image path	perceptual hash of image	product description	unique product code

Table 1: Training Data

The training data is a group of images and text labels. The objective is to learn a model which can group similar

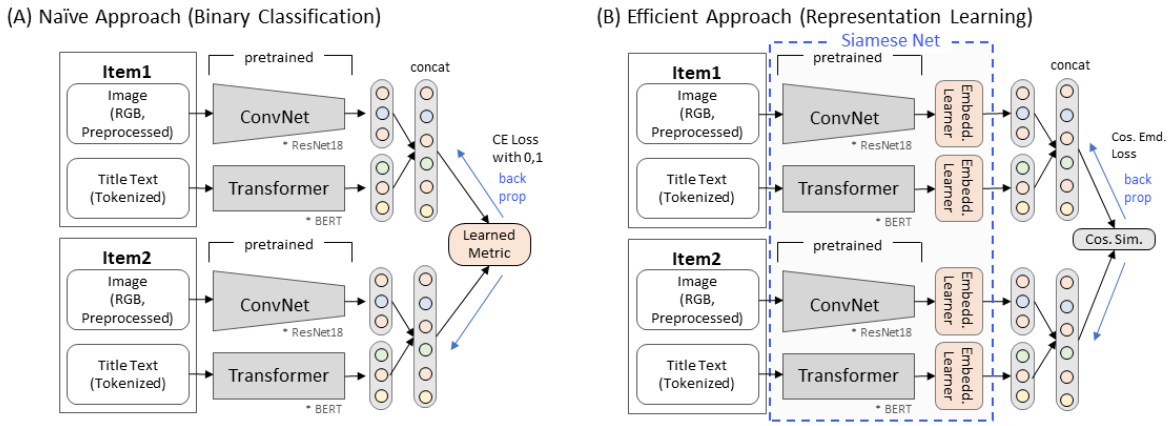

Figure 1: Comparison of Network A and B; Training of Neural Network occurs in module of highlighted in Orange

products. There are tens of thousands of images and texts and there could be thousands of classes in training and test data. Moreover, test data might have images and text which are completely different and ground-truth product, which is associated to postings, never seen in training data.

One of the biggest challenges in product matching is that even for the similar products, the text and image description could be completely different. For instance, there are postings where the similar products have different images with matching text descriptions. Also, there are similar products where text descriptions are completely different or in different languages but images are matching, on the other hand. We believe by combining the image and text information together in neural network model, we can maximize probability of correctly matching products.

Each title has a sequence of words like normal sentences with some differences to be noted. First of all, they are broken sentences. Because a title should keep its length short, it is hard to expect grammatical correctness. Also, postings are all about sales, so the titles have product names, product codes, or discount percentage, which are quite rare in normal sentences, eventually making them hard to be tokenized well. Secondly, titles are indifferent to case. Sometimes a whole title is written with uppercase letters and sometimes the other way. This is the reason why we chose the "uncased" BERT pre-trained model for our text processing.

1.3. Related Work

As product matching is quite a concern in the industry, there were researches on this specific topic, including representation learning. For similarity learning, Koch *et al*. [8] proposed a basic image matching Siamese network. Tracz *et al*. [11] approached product matching problem with BERT with triplet loss, and Yang *et al*. [12] studied document matching with Siamese Transformer. Chen *et al*. [3] focused on multimodal embedding with a large-scale pre-trained Transformer.

2. Naive Approach: Reduction to Classification Problem

As we found during preliminary search, one approach to solve the problem is to reduce the problem to binary classification problem, related to Koch's work [8], which focused on recognizing alphabets across the world. This approach is shown as (A) in Fig. 1.

In order to do this, we prepared Siamese network architecture, first proposed by Jane Bromley *et al*. [2]. The Siamese network consists of two identical neural networks as its components and more importantly parameters are shared. The network is fed with two inputs and outputs feature representation (or embedding). The Siamese network has found its applications in image similarity learning and text similarity learning [1]. Also, the implementation of the network is done with PyTorch Deep Learning framework ann Torchvision [10] pre-trained model.

2.1. Dataset

Given 34K of training data (Table 1), there are $posting\_id$, which is ID of each product 'posting' and there are associated $label\_group$, which is the unique 'product' ID. Thus, in hierarchical view, $posting_id$s belong to one $label\_group$. In fact, there are 11K of unique product among total 34K postings, and the number of duplicated product posting per product is ranging from 2 to 51, meaning that there is no product without duplicated postings.

To use the training data in binary classification task, it is possible to build a paired dataset which consists of $[posting\_id_1, posting\_id_2, image_1, image_2, title_1, title_2, relation]$, where each $image_i$ and $title_i$ contains information of product image and text title respectively, and relation is binary label for indicating whether $posting\_id_1$, $posting\_id_2$ are matched. For instance, when posting A,

B and C are associated to product α, three combination pairs of $[A-B]$, $[B-C]$, $[C-A]$ can be created with $relation = 1$

As it is not possible, due to time and GPU resource constraint, to sweep over all combination of pairs, We've set a parameter named num_neg, which decides the number of total non-matching pair in the training set. When $num\_neg = 1$, there are same number of matching and non-matching pairs. However, test dataset is expected to be extremely unbalanced, because most of product pairs will be labeled as non-matching. Thus, as the label distribution of training and testing data set is different, we expect the model tend to classify test pairs in optimistic way due to its biases. Given num_neg, negative posting is randomly sampled from the set of all non-matching postings. We will explore this aspect more on experiment section by adjusting num_neg parameter.

Also, to measure generalization power, 80% of $label\_group$ belongs to train data set and 20% of it for validation data set. In this way, there is no overlapped product between two data set. As we don't have test data set at our hand (constraint of Kaggle code competition), this implementation is important to keep track of model performance.

2.2. Siamese Network: Embedding

As the problem is reduced to do binary classification, it is quite intuitive that the model can accept two inputs to measure similarity. Siamese is one such network which takes two inputs and outputs a similarity measure between the two inputs.

To figure out this problem, we took the Siamese network as our main hypothesis. Siamese network takes two different inputs and tries to classify whether they have identical class or not with shared model and its parameter. To be specific, we used transfer learning[4] with image feature extractor and text embedding model for the shared engine for Siamese network.

For image feature extractor, we need to compare two images and decide whether they are similar or not. There are existing image classification models which are very good at doing downstream task. Though ResNet is originally designed for image classification problem, but its CNN output extract good image features representation. We have chosen ResNet18[10] model because of its good accuracy and its handling of vanishing gradients issue, in case of fine-tuning the entire model.

For text data from posting titles, we utilized the BERT model and its pre-trained parameters[6] to convert each raw title into a vector representation. The BERT model is well known for its power to represent diverse semantic relationships between words in a sentence into a vector space. Although there are more complex and heavier models than this basic BERT model, even fine-tuning BERT parameters is practically difficult due to limited GPU resources. Because the BERT model takes one (or two) sentence(s) as its input, posting titles are appropriate to be processed without any further aggregation or overly masking out. For a vector representation for each title, we used the average across final transformer block outputs.

2.3. Fusion Issue

The very first issue was how to fuse two very different data sources, images and title texts from each item posting. To make the model simple, we used the most intuitive approach, concatenation. One concern is that the size of output dimension of image and text embedding are different, respectively 512 and 768. Thus, it is expected that title information would take more weight on representation. In the later section of experiment, we extensively address issues of concatenating embeddings.

2.4. Metric: Binary Classification Module

$$D_i = X_i^1 - X_i^2 \tag{1}$$

After producing final embedding vector of each posting X^1 and X^2, distance vector D is defined over Eq. 1. This distance metric can be summed, namely to L1 distance, and eventually used for binary classification of matching and non-matching. However, during the reduction, we will lose some information and weight of features in the final embedding vectors is considered to be same. As we think among extracted features, there must be irrelevant features to do the downstream task, classification.

Thus, we chose to add a classification module to learn the metric itself. While making it simple but have non-linearity to model complexity, we implemented 2 Fully-Connected layer with ReLU activation function. Having the distance vector D as input, it predicts score for binary classification, and then Cross-entropy loss is calculated with given label of $y = [0, 1]$.

2.5. Problem

However, as mentioned earlier, given N total number of postings, there are $\frac{N(N-1)}{2}$ combination for binary classification test, which resulting $O(N^2)$ time complexity.

As the objective is to have up to the 50 best matches for unseen test set, unlike plain image classification, it is required to compare each anchor image with every other image in the test set. Given that the size of test set is expected to be twice of training set (70K), we realized that product matching with binary classification is not possible in practical e-Commerce setting, where the number of product categories keeps increasing.

3. Efficient Approach: Representation Learning

As we discussed above, the Naive approach of using binary classification is computationally complex and practically intractable. Therefore, we turned into more efficient approach, namely learning representation of product posting in vector space. With this change, our inference step needs to be altered as well. As shown above, the naive binary classification approach, requires the model to go through $O(N^2)$ complexity with all possible pairs of postings. However, With representation learning approach, all combination of binary classification pair don't need to be computed but we only need to extract representation once, $O(1)$, and compare pairwise distance between representation. Although pairwise distance comparisons, which is done by hand-coded metric, is also $O(N^2)$ intuitively, we can guess that distance calculation requires much less computation than doing forward pass of deep neural network binary classification with a large number of parameters.

3.1. Non-Linear Embedding Learner

Even with the altered structure of Siamese network, the overall training structure is similar to the previous approach. Binary classification is done as surrogate task to achieve meaningful embedding of product posting. Objective of altered network is to have vector representations worth to call as "embeddings".

To be specific, we don't train the metric itself but train *intermediate* vector representation, precisely embedding learner from (B) in Fig. 1, for image and text separately. We expect the neural network model strive to find right parameter for these learner in order to reduce the loss.

In other words, to make the model reflect relative similarities among product posting, we didn't put no more parameterized model thereafter but just a hand-coded loss. Also, We picked the last output of our fully connected layer as our target embedding vector, having dimension of 512 for both image and text.

One more important aspect is backup from pre-trained models. In Fig. 1, intermediate vectors for image and text are generated from ResNet18 and BERT, respectively. Considering the fact that the model should be robust even to the unseen categories of posting items, "common sense" knowledge will be a great help, as ResNet18 and BERT are well known for their pre-training performance

We have researched several ways to predict the most probable items to be in the same unique product group. One of the most straightforward inference method is k-Nearest Neighbors (k-NN) with threshold and its detail will be discussed thoroughly.

3.2. Metric: Cosine Similarity

Cosine similarity, defined as following Eq. 2, is straightforward similarity measures. It measures the angle between two vectors with normalization to have range of -1 to 1. Specifically, if the cosine similarity is -1, the vector of one is completely opposite of one another, and vice versa.

$$cos(X^1, X^2) = \frac{X^1 \cdot X^2}{\|X^1\|\|X^2\|} \tag{2}$$

$$loss(X^1, X^2) = \left\{ \begin{array}{ll} 1 - cos(X^1, X^2), & \text{if } y = 1 \\ max(0, cos(X^1, X^2) - margin), & \text{if } y = -1 \end{array} \right\} \tag{3}$$

Based on cosine similarity value, we planned to train Siamese network model with cosine embedding loss (Eq.3). This loss function is defined over two embedding vectors of X^1 and X^2 and additional ground truth label y. Unlike previous approach where we used ground truth label of $y = [0, 1]$, we slightly adjust it as $y = [-1, 1]$, where -1 is indicating non-matching relationship and 1 is matching.

Even with training, it is not possible to completely make one's cosine similarity to either -1 or 1. Therefore, hyperparamerter $margin$ will set a threshold to decide how much close is enough to decide whether two product posting is matching or not. For instance, given matching posting pairs, if cosine similarity of is 1, then the loss will be 0, meaning that there is no space for model to improve its performance. On the other hand, with regard to non-matching posting pairs, loss is determined as difference between cosine similarity and $margin$. If resulted embedding has $cos = 0.7$ for a non-matching pair given $margin = 0.5$, then the model is penalized by 0.2.

With regard to $margin$, we can interpret it as a penalty mitigation term for negative pairs. This is the only adjustable parameter we have in our hand and has a direct impact in representation training. If we set the value of $margin$ too high, it won't provides sufficient amount of feedback to the network to make different items distant to each other. If too low, the output embedding will get too sparse and might lose continuity in learned manifold. This means that the embedding network will be less robust to the unseen item classes.

Intuitively, $margin$ can be used as classification standard for binary case. Because loss and cosine similarity itself is not from hard prediction unlike Cross-entropy, we used $margin$ as a threshold for binary classification in order to measure model performance. In particular, when cosine similarity is higher than $margin$, we regard the model predicted the pair as matching, and vice versa. As this term affected the model performance significantly during the training, we will investigate this hyperparameter in Experiment section extensively in both quantitative and qualitative manner.

3.3. Finding the match

Training is done to find best image and text embedding from the model. Once training is completed then output of image and text embedding for test data is used to find up to 50 nearest matches. We considered two options for finding nearest matches – Facebook AI Similarity Search (FAISS) and k-nearest neighbors (k-NN) with Scikit-learn. FAISS is a similarity search library developed by Facebook. This library is based on multiple fast search methods - product quantization, three-level quantization, inverted multi-index, optimized product quantization and many more. FAISS supports both Euclidean and cosine distances. FAISS support similarity search on GPU. FAISS is useful for very large number of vectors which could not fit in RAM. Scikit-learn's k-NN uses kd-Tree and ball tree for fast search on the nearest matches. We have implemented similarity search using both FAISS [5] and Scikit-learn's k-NN library [9]. To find up to 50 matches, first, all of the embedding, which is generated by trained representation learner, of test data is fed to k-NN model. The k-NN model can be requested to return the number of neighbours and their distances from the query data point. As per the problem statement we must find up to the 50 best matches. Therefore, We first get 50 nearest matches from the query data point and then we experimented to find a distance threshold to find only matching products.

4. Experiments and Results

4.1. Measurement of Success

There are two phases; learning embedding with the surrogate task and product matching phase. During training, binary classification done for learning, and product matching is done for multi-class multi-label output without more training. Therefore, there are two different evaluation metrics used in each phase.

4.1.1 Evaluation Metric for Surrogate Task

For both training and test task, binary classification and product matching, accuracy was not meaningful metric for model performance. With the case of binary classification, many of posting pairs would be labeled as non-matching. Specifically, there is a dedicated hyperparameter, num_neg which decides ratio of matching and non-matching label in the training data set. Whem num_neg is high, this sense, a pessimistic model, which tends to classify pairs as non-matching would score better than others, due to large True-Negatives. On the contrary, F1 score can be defined as $\frac{TruePos}{TruePos+\frac{1}{2}*(FalsePos+FalseNeg)}$. As this metric lacks $TrueNegative$, which is too prevalent in both training and test task, we can properly measure model performance. This is proven that direction of F1 and cosine embedding loss is well-aligned during training phase.

4.1.2 Evaluation Metric for Product Matching

The objective of product matching task is to produce all matching $posting\_id$, given $posting\_id$ of the query, making it as multi-label learning. The biggest challenge comes from the fact that as the number of labels increases, the number of possible label sets increases exponentially. For example, in this competition there are up to 50 labels for a query, therefore there are 2^{50} label sets are possible. In conventional supervised learning where one class has one label, there are several metrics to evaluate the performance of a learning task – accuracy, F1-score, area under the ROC curve. These metrics cannot be applied to multi-label task in a straightforward manner. As per the competition requirement, submissions will be evaluated based on mean F1 score of each product posting. The mean is calculated in a sample-wise fashion, meaning that an F1 score is calculated for every predicted row, then averaged [7]. This methodology is an example-based metric [13], so that the precision of a sample is the ratio of how many prediction are correct to the total number of prediction for an input query. The mean value over all queries is taken as the precision value of complete set.

$$Precision = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap h(x_i)|}{|h(x_i)|}$$

Where, X is example space, n is total number of samples, x is feature vector in X, Y is label set associated with the x, $h(.)$ is multi-label classifier. The recall of a sample is the ratio of how many prediction are correct to the total number of ground truth for an input query. The mean value over all queries is taken as the recall value of complete set.

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap h(x_i)|}{|Y_i|}$$

The F1 scores is calculated from $precision$ and $recall$ as following.

$$F1\_score = \frac{2 * Precision * Recall}{Precision + Recall}$$

We have used above metrics for product matching performance measurement.

4.2. Model Training

At first, we tried to extract *universal* feature representation from image and text data just using pre-trained model. However, as noted in [14], without task-specific training, the expressive power from pre-trained model is limited as

Hyperparameter		Concatenated				Text				Image			
$margin$	num_neg	Loss	F1	Prec.	Rec.	Loss	F1	Prec.	Rec.	Loss	F1	Prec.	Rec.
0.7	10	0.0317	0.7152	0.5860	0.9173	0.0314	0.7079	0.5904	0.8838	0.0338	0.6762	0.5672	0.8372
	5	0.0536	0.7454	0.6139	0.9487	0.0536	0.7321	0.6111	0.9130	0.0559	0.7220	0.6016	0.9025
	1	0.1150	0.8146	0.6952	0.9836	0.1228	0.8100	0.6897	0.9811	0.1154	0.8165	0.6986	0.9823
0.5	10	0.0193	0.6936	0.5593	0.9129	0.0190	0.6687	0.5344	0.8932	0.0200	0.6700	0.5463	0.8664
	5	0.0328	0.7239	0.5884	0.9405	0.0324	0.7147	0.5845	0.9198	0.0334	0.7142	0.5848	0.9173
	1	0.0722	0.8045	0.6787	0.9874	0.0758	0.7980	0.6707	0.9851	0.0709	0.8074	0.6823	0.9888
0	10	0.0906	0.0156	0.6384	0.0079	0.9091	0.1667	0.0909	1.0000	0.0896	0.0370	0.8163	0.0189
	5	0.8333	0.2857	0.1667	1.0000	0.1631	0.0627	0.7421	0.0327	0.1629	0.0634	0.7608	0.0331
	1	0.2439	0.7660	0.7080	0.8342	0.4995	0.6667	0.5000	1.0000	0.2803	0.7094	0.7816	0.6494

Table 2: Snapshot of *training* result at 10th epoch for binary classification task with prepared paired data set (learning rate of 0.001). There are three categories of information used Text+Image, Text, and Image, where Text+Image showed higher performance in $num\_neg = 10$ setting. Siamese network from Fig. 1 of (B) is used for representation learning

seen in the Fig. 2. During the training of binary classification task, we mainly tuned following hyperparameters; $num\_neg = [1, 5, 10]$, $margin = [0, 0.5, 0.7]$. Those are determining factor for composition of training data set and directly related to model's biases.

For the implementation of training and transfer learning, our code is based on PyTorch Transfer Learning Tutorial[4], and added features such as cached dataloader. For optimization, we used Stochastic Gradient Descent ($momentum = 0.9$) and decaying learning rate by every 7th epoch by 0.1, which is as same as the reference. For preprocessing of image, we followed the same transformation of reference as well, because pre-tranined model is trained in such way. Also, due to resource constraints, and in order to consistently compare above focused hyperparameters, $batch\_size$ and $epoch$ is fixed for training phase as 128 and 10.

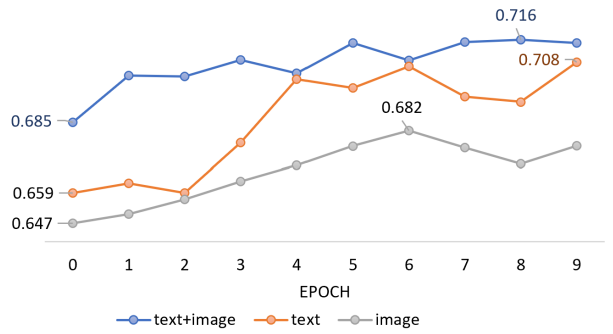

Figure 2: Learning curve of binary classification task, given different information source ($margin = 0.5$, $num\_neg = 10$)

One major issue is that concatenation might result improper training. This improper training happens when image and text embedding don't agree to each other. For example, given a matching pair, say cosine similarity of image embedding is near to 1, and text embedding's similarity is close to -1. When the model incorrectly predicted them as non-matching, backward gradient of loss function would penalize image embedding model as well, although image embedding's representation was good enough. Due to this drawback, training the concatenated model is found to be unstable. The origin of this problem is, the model only can correctly learn from when they are correctly aligned. In order to make the training more stable, I increased batch size from 32 to 128. Although simple, but it reduced instability in training process.

Learning rate is selected from $[0.1, 0.01, 0.001]$, but due to instability of training as mentioned above, the model only can be properly trained with 0.001. With 0.1, the weight of neural network diverged, and also with learning rate of 0.01, loss and F1 score degraded as epoch passes by.

Also, another interesting hyperparameter was $margin$. At first, we thought that $margin$ doesn't affect model performance after proper training. This is because, as we added non-linear transformation layer, referred as embedding learner, it is expected that the layer can have sufficient model capacity to completely transform embedding of pre-trained model. This hypothesis was also supported by the definition of cosine embedding loss in Eq. 3, in which smaller margin produces stronger signal. However, it turned out that even with $margin = 0$, there is bottom line that the layer cannot transform further. Opposed to our first assumption, the model is sensitive to $margin$. For instance, with $margin = 0.7$, text-only model severely suffered after epoch 5, while text+image model is rather unaffected or even slightly improved. This evidence also support our motivation at the first, combination of text and image information could lead robust and accurate model.

With regard to model overfitting, the model showed some degree of overfitting. One interesting aspect is that overfitting occur mainly on image and $concat$(text+image), while text embedding learner doesn't show much overfitting.

4.3. t-SNE Visualization

To be aligned with our purpose, we need to check if the final output vectors, which are supposed to be "embeddings", show satisfactory mapping results. Here, we want

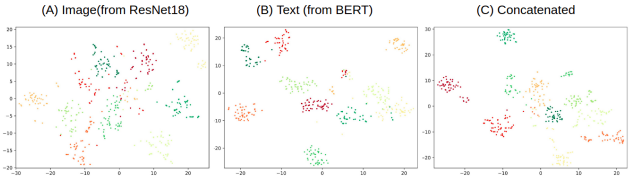

Figure 3: t-SNE Visualization of Ten $label\_group$ without Training

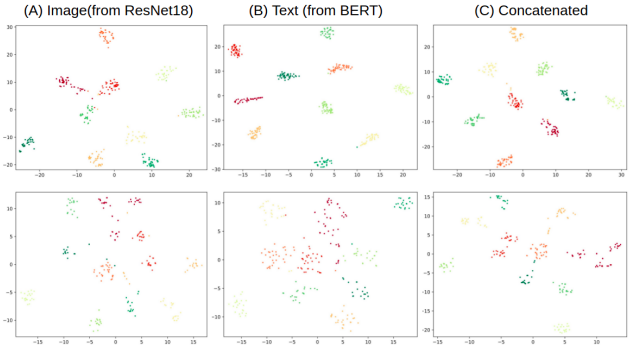

Figure 4: t-SNE Visualization of Ten $label\_group$ after Training (First Row: Train, Second Row: Validation)

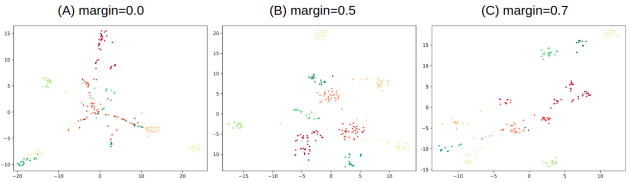

Figure 5: t-SNE Visualization of Ten $label\_group$ by $margin$ (Trained and Representing Validation Set)

to present visual mappings with the help of dimension reduction technique.

Because usually embedding vectors have fairly high number of dimensions in our case 512 for both image and text, we need to project onto 2 or 3 dimensional space for intuitive analysis. Among many dimension reduction methodologies, t-Distributed Stochastic Neighbor Embedding (t-SNE) is well known for visualization purpose because of its stochastic property and robustness with respect to its parameters including perplexity.

Representations from pre-trained ResNet18, BERT, and concatenated, namely without training at all, are shown in Fig. 3. In terms of intra-cluster($label_group$) distance, even without any training, local clusters of posting item have been formed, providing evidence that pre-trained model's common sense knowledge worked. Also, with regards to inter-cluster distance, BERT and concatenated have shown decent separation between $label_groups$. What's worth to note is that image embedding has shown poor result in both inter/intra-cluster distances, which might be the reason why image embedding is not prevalent as text embedding.

In order to gauge effect of training on binary classification task, we compared Fig. 3 with Fig. 4's first row. What's inspiring is, regardless of image, text or concatenated, training improved both intra and inter cluster distance, namely intra-group cohesiveness and inter-group separation.

By comparing representation on first row (on training data) to second row (on validation data) in Fig. 4, we can measure the embedding network's generalization power. Concatenation showed better generalization by retaining intra-cluster distance, and BERT has shown the weakest generalization ability, which is surprising as to BERT's success in Fig. 3. It seems that training made the BERT representation overfitting to the training data.

Results from t-SNE with different $margin$s in Fig. 5, showed a notable result. With $margin = 0$, default value for nn.CosineEmbeddingLoss in PyTorch, trained representation vectors are aligned along few lines, showing poor result. It was required to tune $margin$ to properly train the model. But why?

To reason about it for our case, posting items have many label to be matched, around 9K in training data, constructing somewhat continuous category space. For example, a watch and a smart watch can be categorized differently, but intuitively we hope their embeddings are fairly close so that it can represent their similar aspects well. With binary classification surrogate task with $margin = 0$, however, the network is deprived of any chance to consider their potential similarity, but just makes them far apart. This might have lead the learner to fail to represent postings appropriately, and this is why giving some "$margin$" to the loss can work better in sometimes. For $margin = [0.5, 0.7]$ cases, both have resulted decent level of clustering.

From above, we've got three lessons. First, using pre-trained models for image and text was surely helpful with their common knowledge. Also, it is found that training with surrogate task can improve quality of embeddings. Lastly, concatenated representation showed better generalization and clustering property than image or text only case.

4.4. Result of Product Matching

The training data provide us only matching labels, moreover, after data analysis we found that there are 2 to 51 instances belong to a label group. Therefore, there are not many examples for a class like a conventional supervised learning environment. Consequently, we had prepared our own negative examples pairing two products, sampling num_neg negative pairs from non-matching set of postings. During training, we had used binary classification to compare two products. But the real product matching environment is completely different. We have to choose the matches out of all product instances for a query. After the training is completed using positive and negative examples, we used final embedding to find the optimum distance

threshold over our training data to distinguish matching and non-matching products.

Thr	Image	Thr	Text	Thr	Image+Text
0.87	0.6352	0.77	0.5816	0.78	0.6605
0.88	0.6526	0.78	0.5869	0.79	0.6656
0.89	0.6594	0.79	0.5895	0.80	0.6671
0.9	0.6607	0.80	0.5897	0.81	0.6672
0.91	0.6575	0.81	0.5878	0.82	0.6633
0.92	0.6512	0.82	0.5859	0.83	0.6579
0.93	0.6432	0.83	0.5823	0.84	0.6511
0.94	0.6345	0.84	0.5778	0.85	0.6423

Table 3: Training Data

Thr	Image	Thr	Text	Thr	Image+Text
0.9	0.6802	0.80	0.5951	0.81	0.6768

Table 4: Validation Data

Text and image have different optimal threshold, it means that their embedding is placed differently in vector space. The optimal threshold value obtained from training is used in validation data and it is observed that validation F1-score is better than training score. This can be explained by the fact that training data has 27,615 samples while validation data has only 6,635 samples, therefore it would be easier for the model to match.

The surrogate task of learning embedding through binary classification has shown that concatenated embedding resulted in better performance, compared to text/image only. When we applied image and text combined embedding for final product matching, we found that image+text performance is as same level of matching image only performance. In our opinion, reason for it might be that number of negative matches are huge because kNN is searching in complete validation set while surrogate task is done over 10 negative examples. We are hopeful that increasing number of negatives in surrogate task training would help to get superior results in final product matching. This is left as future work to be done. As Kaggle competition is still open, we will push it forward.

5. Conclusion

While it is widespread convention to use embedding in NLP related task, such usage of embedding in computer vision task, like in this case, image similarity is restricted. This is because the unit of natural language data is apparently defined as word, while the unit of computer vision is unclear, of course pixel should be a unit. However, according to result from binary classification and product matching task, the concatenated representation vector showed decent performance.

As concatenation will definitely increase complexity of model, we concerned with curse of dimentionality. In many machine learning application, adding more information doesn't necessarily bring performance improvement. However, our simple concatenation technique worked despite of increased complexity. One reason of this success would be because the source of image and text information is independent to each other, although it is indeed originated from one real product.

One thing to note is that we've applied various techniques on this project; transfer learning, feature extraction, surrogate task, Siamese network, t-SNE and k-NN.

6. Discussion

While we have succeeded to train model with concatenation, the origin of instability training still remains, wrong loss signal will be given to models when two embeddings disagree. To improve further, loss should penalize only embedding model which is indicating wrong direction.

One simple solution to the problem would be using PyTorch's dynamic computational graph feature. Unlike TensorFlow, PyTorch allows to modify network structure during training or evaluation. Thus, we can catch ground truth label information and let the loss gradient flow to a certain embedding model. Besides this framework-dependent solution, we can train embedding models separately and only concatenate them in validation and test phase.

Additionally, we can use triplet loss for training of Siamese network. Unlike loss functions such as cosine embedding loss which measures distance between two vectors, this loss function is defined over triplet vectors (Anchor, Positive and Negative) as in Eq. 4.

$$loss(A, P, N) = max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0) \quad (4)$$

One advantage of using triplet loss is that it is possible for the model to clearly discern positive and negative embedding, as both distance $dist(A, P)$ and $dist(A, N)$ are maximized at the same time. Because combination of three vectors will be much larger than combination of two, it is required to plan better sample strategy.

7. Work Division

To see each member's contribution to this project, please refer to Table 5.

References

[1] bhilash Nandy, Sushovan Haldar, Subhashis Banerjee, and Sushmita Mitra. A survey on applications of siamese neural networks in computer vision. *International Conference for Emerging Technology (INCET)*, 2020. 2

[2] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *IJPRAI*, 7(4):669–688, 1993. 2

[3] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *European Conference on Computer Vision*, pages 104–120. Springer, 2020. 2

[4] Sasank Chilamkurthyh. Transfer learning for computer vision tutorial. 3, 6

[5] Facebook. Facebook faiss libarary: Getting started. https://github.com/facebookresearch/faiss/wiki/Getting-started. 5

[6] huggingface. Pre-trained bert nlp model. https://huggingface.co/transformers/model_doc/bert.html. 3

[7] Kaggle. Evaluation: Shopee competition. https://www.kaggle.com/c/shopee-product-matching/overview/evaluation. 5

[8] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015. 2

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 5

[10] PyTorch. Pre-trained computer vison model. https://pytorch.org/vision/stable/models.html. 2, 3

[11] Janusz Tracz, Piotr Iwo Wójcik, Kalina Jasinska-Kobus, Riccardo Belluzzo, Robert Mroczkowski, and Ireneusz Gawlik. BERT-based similarity learning for product matching. In *Proceedings of Workshop on Natural Language Processing in E-Commerce*, pages 66–75, Barcelona, Spain, Dec. 2020. Association for Computational Linguistics. 1, 2

[12] Liu Yang, Mingyang Zhang, Cheng Li, Michael Bendersky, and Marc Najork. Beyond 512 tokens: Siamese multi-depth transformer-based hierarchical encoder for document matching. *arXiv preprint arXiv:2004.12297*, 2020. 2

[13] Zhi-Hua Zhou and Min-Ling Zhang. *Multi-label Learning*, pages 875–881. Springer US, Boston, MA, 2017. 5

[14] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin D. Cubuk, and Quoc V. Le. Rethinking pre-training and self-training, 2020. 5

Student Name	Contributed Aspects	Details
Ash	Simliarity Searching, Image embedding models, evaluation metric, product matching, Kaggle Notebook submission, validation set preparation, validation	Implemented k-NN and FAISS similarity search. Implemented output label generation for a query. Created Kaggle Notebook for submission. Created validation set for testing. Prepared image embedding from pretrained model and generated product matching label. Generated product matching labels from image embedding, text embedding and image+text embedding. Implemented precision, recall and f1-score for multi-label classification. Research on product matching approaches. Researched on Siamese network.
Chungjin Lee	Theoretical Research, Data preparation, Model Implementation, Model Training and Hyperparameter Tuning, Model Evaluation	Researched on similarity matching with regard to model and loss function. Prepared paired data set for training purpose. Implemented data pipeline (PyTorch custom class) and optimized its performance (caching). Implemented Siamese Net architecture. Trained image/text representation learner (on binary classification task). Evaluated effect of hyperparameters and training performance (metrics, learning curve)
Jungho	Research, Embedding Design, Model Implementation/Integration, Visualization	Searched intensively about embedding vector representations including its meaning, property, methodology and how to design the network. BERT handling for text part and its integration to the main framework. Qualitative interpretations with t-SNE figures.

Table 5: Contributions of team members