CPE 646 Project

Spring (2021)

Clustering Similar items across different sources based on Title and Images.

Team members –

- Ankur Chaudhari (10468473)
- Sudipto Sen (10458905)

Introduction

Goal: To build a model that compares items to predict if they are the same product.

Problem Description: We often find retail companies offering recommendations in which they promote their products in such a way that customers tend to get swayed and pick a similar product that is priced lower. Product matching is one of these strategies wherein a company offers products at rates that are competitive to the same product sold by another retailer.

These matches can be performed automatically with the help of Machine Learning algorithms, and we intend to use an ensemble of several algorithms to get the best accuracy. We have been provided with data of Shopee, which is the leading e-commerce platform in Southeast Asia and Taiwan.

No one likes paying extra for the same product depending on where they shop. Retail companies use a variety of methods to assure customers that their products are the cheapest. To perform these matches automatically requires a thorough machine learning approach, which is where this project can be beneficial.

Two different images of similar wares may represent the same product or two completely different items. Retailers want to avoid misrepresentations and other issues that could come from conflating two dissimilar products. Currently, a combination of deep learning and traditional machine learning analyzes image and text information to compare similarity. But major differences in images, titles, and product descriptions prevent these methods from being entirely effective.

Data Description

Finding near-duplicates in large datasets is an important problem for many online businesses. In Shopee's case, everyday users can upload their own images and write their own product descriptions, adding an extra layer of challenge. Our task is to identify which products have been posted repeatedly. The differences between related products may be subtle while photos of identical products may be wildly different!

Files

- [train/test].csv - the training set metadata. Each row contains the data for a single posting. Multiple postings might have the exact same image ID, but with different titles or vice versa.
  - posting_id - the ID code for the posting.
  - image - the image id/md5sum.
  - image_phash - a perceptual hash of the image.
  - title - the product description for the posting.
  - label_group - ID code for all postings that map to the same product. Not provided for the test set.

❖ [train/test] images - the images associated with the postings.

❖ sample_submission.csv - a sample submission file in the correct format.

matches - Space delimited list of all posting IDs that match this posting. Posts always self-match. Group sizes were capped at 50, so there is no need to predict more than 50 matches.

Evaluation metric: F1-Score

The evaluation metric for this competition is F1-Score or F-Score.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

$$\textbf{Precision} = \frac{tp}{tp + fp}$$

$$\textbf{Recall} = \frac{tp}{tp + fn}$$

It finds the balance between precision and recall.

where-

- TP = True Positive
- FP = False Positive
- TN = True Negative
- FN = False Negative

Data Analysis

Our primary aim is to classify images based on embeddings and to find out the images of the same product from dissimilar sources using an ensemble of different models. Thus, we begin by analyzing the data provided before we begin processing it.

With the hope of finding patterns or trends in data, we first represent the data in the form of several graphs. Thus, we are plot the images label wise and title wise. We also plotted random images, images of specific labels, as well as images of specific title.

We use bar graphs for plotting image count, and images in the same label group. Thus, we find the top five products. We have also created word-cloud based on image title just to find out which words appear most frequently.

After performing Natural Language Pre-processing techniques such as removing stop-words, converting characters to lowercase and removing punctuation, performing tokenization, stemming and lemmatization on title attribute of the data, we used plotly library for plotting title length distribution, word count distribution and character count distribution in order to get the gist of title parameter of dataset and to check whether such parameters have any particular distribution in order to use them for model development.

Data Processing

As we are dealing with 32400 image files, we decided to use GPU processing for faster implementation of project. Rapids.ai provides us with solution to use GPU for data processing along with easy python integration in order to reduce training time using scaling data on GPUs.

In order to use RAPIDS as well as tensorflow, we restricted tensorflow to 2GB of GPU RAM and we allocated 14GB of GPU RAM for RAPIDS data processing.

Model 1

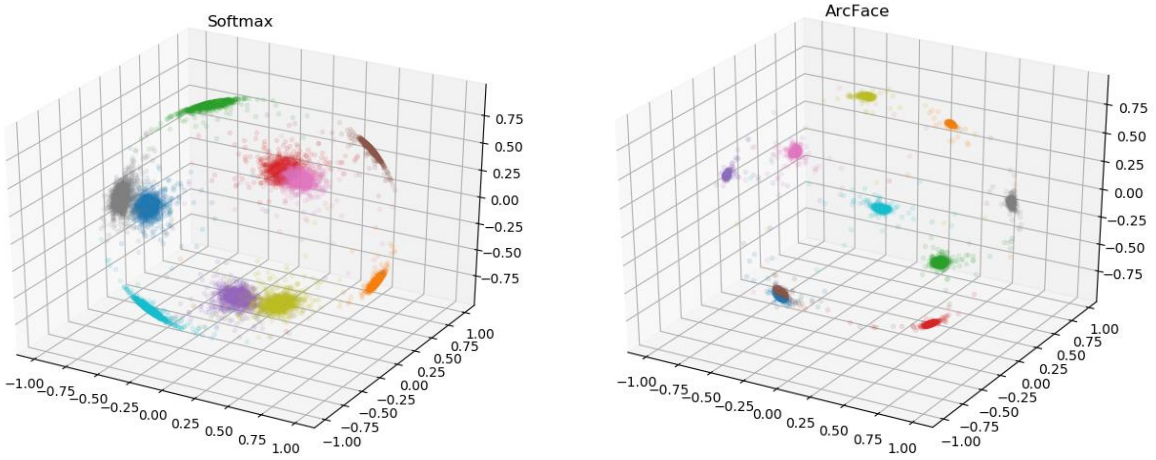
ArcFace

To find text and image embeddings, we initially define ArcMarginProduct function using efficientNetB3 for getting image embedding. Thus, this function is used to get the embeddings of our images with the fine-tuned model.

This can be used for implementing large margin arc distance. If we normalize every embedding (i.e., divide each embedding by its magnitude), then all the embeddings reside on a unit sphere in D-dimensional space where D is the size of our embedding.

ArcFace adds more loss to the training procedure to encourage similar class embeddings to be close and dissimilar embeddings to be far from each other. This is adding a second task to the training of a CNN. The first task is predicting the image accurately.

Now, we intend to use ArcFace Embeddings. If we train with ArcFace loss then while we plot the embeddings, we see that similar classes are close together and dissimilar classes are farther apart.
Below are representation of softmax embeddings without ArcFace and ArcFace embeddings.



Tuning ArcFace

It was the first time for both of us to use ArcFace, so at first we had a hard time tuning it. A sufficiently large margin was important for the quality of the embedding, but when we increased it, the model suffered from convergence issue. We found several ways to overcome this issue, and finally applied the following options.

- increase margin gradually while training.
- use large warmup steps.
- use larger learning rate for cosinehead.
- use gradient clipping.

Margin of 0.8~1.0 was optimal for image model, and 0.6~0.8 for text model. When using the increase-margin approach, we start from a margin of 0.2 and increase it to 1.0 for image model and 0.8 for text model.

Also, we tried class-size-adaptive margin. It is best when margin is set to class_size^-0.1 for image model and class_size^-0.2 for text model. However, the improvement was subtle since the class was less imbalanced.

Using larger learning rate for cosine-head and using gradient clipping not only made the model converge, but also improved CV score a bit. Adding extra fc layers after global average pooling hurts the model performance, while adding batch normalization before feature-wise normalization improved the score.

Embeddings

Imagine that we want to compare two images and decide whether they are similar. Images are hard to compare, but numbers are easy to compare. So, we input an image into a CNN and take the activations of the last layer before output layer. So, we can input two images, get two embeddings, and then compare the embeddings. The CNN embeddings are meaningful because they represent patterns that are detected in the images.

For text title embedding we have used the pre-tuned model – bert-en-uncased Model, which is a model for getting text embedding. It returns tokens, masks and segments from text array.

We used Bert based Text Encoder and Image Encoders with different backbones (mainly nfnet,and effnet). As many other teams in this kaggle competition, we did see some improvement in single models when tuning the training and the backbone, but in the blend this effect was almost vanishingly small and we were kind of reaching a plateau there. Our embeddings usually had a length around 1024 with larger embeddings yielding a slightly better score on CV/LB.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inp1 (InputLayer)            [(None, 512, 512, 3)]     0
_________________________________________________________________
efficientnet-b3 (Functional) (None, None, None, 1536)  10783528
_________________________________________________________________
global_average_pooling2d (Gl (None, 1536)              0
=================================================================
Total params: 10,783,528
Trainable params: 10,696,232
Non-trainable params: 87,296
_________________________________________________________________
None
Our image embeddings shape is (34250, 1536)
Model: "model_3"
__________________________________________________________________________________
____________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
====================
input_word_ids (InputLayer)     [(None, 70)]         0
__________________________________________________________________________________
____________________
input_mask (InputLayer)         [(None, 70)]         0
__________________________________________________________________________________
____________________
segment_ids (InputLayer)        [(None, 70)]         0
__________________________________________________________________________________
____________________
keras_layer (KerasLayer)        [(None, 1024), (None 335141889   input_word_id
s[0][0]
                                                                 input_mask[0]
[0]
                                                                 segment_ids
[0][0]
__________________________________________________________________________________
____________________
tf.__operators__.getitem (Slici (None, 1024)         0           keras_layer
[0][1]
==================================================================================
====================
Total params: 335,141,889
Trainable params: 335,141,888
Non-trainable params: 1
__________________________________________________________________________________
____________________
None
Our text embeddings shape is (34250, 1024)


370597
```

KNN

Finally, we use KNN model using cosine metric to get 50 nearest neighbors for an image provided, making different cross validation sets for finding out the F1 score for every model with different threshold values. In order to get threshold values for text embedding as well as image embeddings, we ran this model structure in interactive mode, then replaced else clause with a solid threshold. We then use the best F1 score, for best accuracy. In case of confidence parameter for KNN model, we are predicting the test set that have 70K images and different label groups so ideally confidence should be smaller.

TFIDF – text embeddings

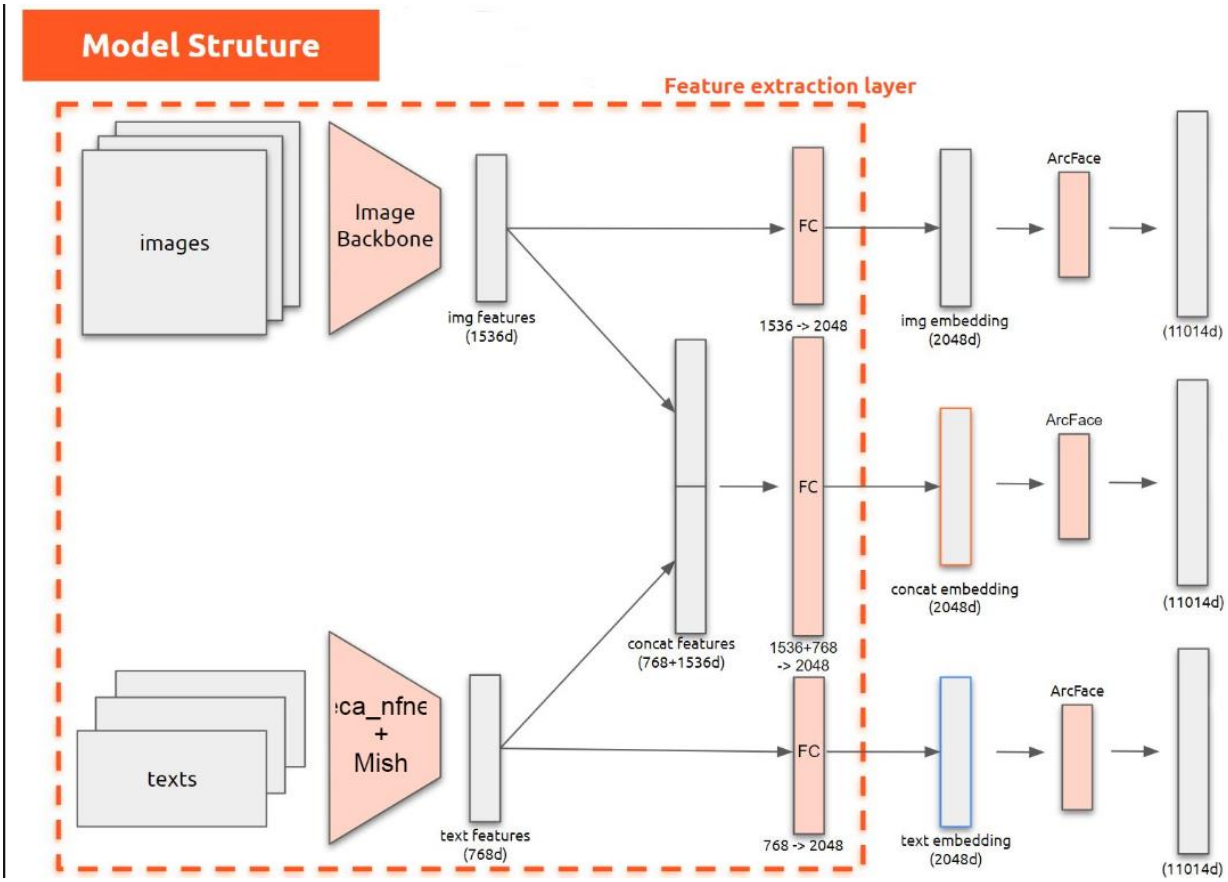
TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

We added TF-IDF (char and word) to the mix of embeddings and even though CV did not change much it seemed to help a few points on LB.

Cosine Similarity

We used cosine similarity for finding similar title. For using cosine similarity, we need to introduce cosine distance.

Cosine Distance - We compare vectors (numbers) by computing the distance between them. For example, we can find the distance between the 3-dimensional vector [0.2, 0.9, 0.7] and [0.5, 0.4, 0.1]. The cosine distance is defined as one minus the cosine of the angle from point one to the origin to point two. This equals 0 when the points are the same, and 1 when the points are far away. Thus, when cosine distance is closer to 0, then the vectors have cosine similarity.



Model 2

For our second model, we implemented ArcMarginProduct_Image function with neural network backbones instead of keras layer. In order to get better accuracy we tried several neural networks such as resnext50_32x4d, efficientnet_b3, tf_efficientnet_b5_ns, eca_nfnet_l0 with 2D adaptive average pooling layer, linear layer, dropout and 1D batch normalization. After implementing the models with different parameters, we found eca_nfnet_10 to be best backbone to work with

Mish Function -A Self Regularized Non-Monotonic Activation Function

Mish has a stronger theoretical pedigree, and in testing delivers, on average, superior performance over ReLU in terms of both training stability and accuracy. A simple mathematical definition of Mish is -

Mish = x * tanh(ln(1+e^x)).

For reference, ReLU is x = max(0,x) and Swish is x * sigmoid(x).

Most importantly, current thinking is that smooth activation functions allow for better information propagation deeper into the neural network, and thus better accuracy and generalization.

That being said, We have tested several activation functions that also met many of these ideals, and most failed to perform. The main difference here is likely the smoothness of the Mish function at nearly all points on the curve.

We used Mish activation function along with eca_nfnet_10 pre-trained neural network model using timm and replacing its activation function with Mish in order to get better CV accuracy.

Result

We were successfully able to cluster images of the same product from different sources for test dataset, with an F1- Score = 0.8725 and Kaggle competition score of 0.722 when ran on Hidden Dataset.

Conclusion

In our case, an ensemble of the predictions from different models used is more accurate than predictions made by any of the models when implemented individually. Thus, we can conclude that the different models used within each ensemble for making predictions must have low correlation. Generation and concatenation of Image embeddings as well as text embeddings for getting similar items returns better result than only working with images or texts.

References

https://arxiv.org/pdf/1801.07698.pdf

https://github.com/lyakaap/Landmark2019-1st-and-3rd-Place-Solution/blob/master/src/modeling/metric_learning.py

https://github.com/tyunist/memory_efficient_mish_swish/blob/master/mish.py

https://arxiv.org/pdf/1908.08681.pdf

https://www.kaggle.com/cdeotte/part-2-rapids-tfidfvectorizer-cv-0-700

https://www.kaggle.com/muhammad4hmed/b3-tfidf-knn-boom-p

https://www.kaggle.com/parthdhameliya77/pytorch-eca-nfnet-l0-image-tfidf-inference

https://medium.com/walmartglobaltech/product-matching-in-ecommerce-4f19b6aebaca

https://towardsdatascience.com/unravelling-product-matching-with-ai-1a6ef7bd8614

https://towardsdatascience.com/ml-powered-product-categorization-for-smart-shopping-options-8f10d78e3294

http://www.semantic-web-journal.net/system/files/swj1470.pdf

https://www.aclweb.org/anthology/2020.ecomnlp-1.7.pdf

https://www.researchgate.net/publication/254005851_Tailoring_entity_resolution_for_matching_product_offers

http://data.dws.informatik.uni-mannheim.de/largescaleproductcorpus/data/v2/papers/DI2KG2020_Peeters.pdf

https://www.cs.cornell.edu/~kb/publications/SIG15ProductNet.pdf

http://ceur-ws.org/Vol-2631/paper18.pdf