Product semantic similarity using representation learning

Apoorva Jarmale, Chirag Tagadiya, Yogesh Nizzer

1 Motivation

Shopee is the leading e-commerce platform in Southeast Asia and Taiwan in which users can upload their own images and write their own product descriptions.Customers appreciate it's easy, secure, and fast online shopping experience tailored to their region. Finding near-duplicates in large datasets is an important problem for businesses like Shopee. Retailers want to avoid misrepresentations and other issues that could come from conflating two dissimilar products. Based on this we propose to work on a project where we'll build a model that predicts which items are the same products based on images and text. With robust and useful recommendation systems, users will spend more time on websites and purchase more products, which in turn increases profits. Based on this, we will be building a generalized system for finding semantically similar products based on images and text description.

2 Dataset

We will use the Shopee e-commerce dataset provided in Shopee. The Shopee dataset contains 34250 train images. We expect to find roughly 70,000 images in the hidden test set. Apart from images, we have perceptual hash of the image, and product description for each of the items. We also have "label_group" for each product, label_group is the id code for all the products that map to the same item, It is not provided for test data. We have a very large test dataset, there might be possibilities that some of the label_group present in test data might not be in train data. This makes this problem very challenging as we can not only rely on train label_group.

3 Exploratory Data Analysis

For the image model, we explored the pattern between Image and label_group. We found certain images, those are almost identical, the only difference is **zoom level,background and orientation**. (Figure 1) We explored another set of images, which belong to **different label_group, but they look very similar** (Figure 2). This makes the problem very challenging.

Figure 1

Figure 2

For the text model, we had to explore the text part of the dataset. The columns of interest for this model are posting_id, title and label_group. There are 34,250 rows in dataset with 11,014 unique label groups. There are 33,117 unique titles which means that we can assume that the titles don't really repeat much (as expected). Since we are looking at supervised machine learning algorithms for the text model, we will be considering the label_group as the ground truth since this is true most of the time. The titles in the dataset for each of the images have words from multiple south-east asian languages like malaysian,indonesian etc apart from english as can be seen in Figure 3.

Figure 3

The distribution of the number of words in the titles is as shown in Figure 4. The length is between 0-20.

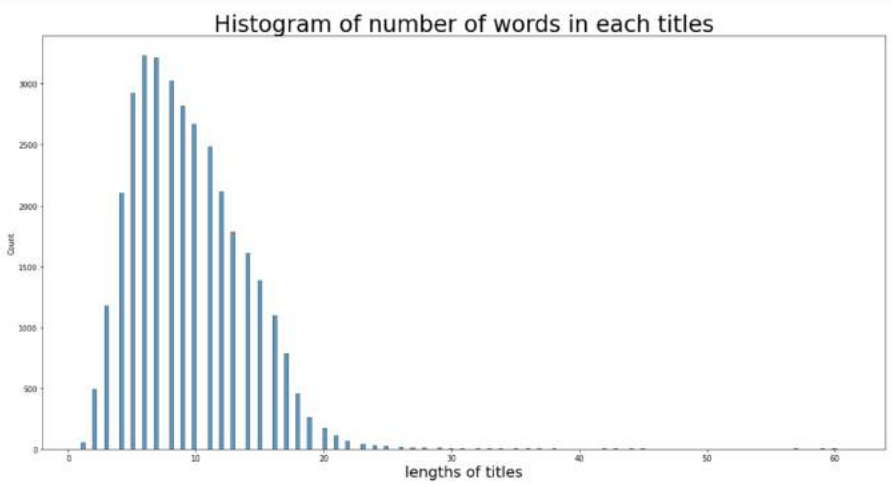


Figure 4

4 High Level Design and Architecture

Take a look at the image below (**Figure 3**), User is thinking about a “**Maroon shirt**” in his mind, so we would like to build a Machine learning system that will recommend two products shown as **TRUE** in the image below, but we don’t want to recommend a **FALSE** image.

We divide the overall solution in the following 3 parts.

1. **Learn to represent objects (Product titles, product images) as a continuous dense vector** (i.e. generate product title embeddings and product image embeddings)
2. **Learn to place similar objects together**. Similar product image should be in neighborhood, same way similar product title should be in same neighborhood. More difficult task would be to place a similar product title and similar product image in the same neighborhood. (Our next step)
3. Learned to **retrieve neighboring** objects and product embeddings **really fast.**

Figure 5

5 Data Modeling

5.1 Image modeling using autoencoder

Auto Encoder consists of two models Encoder and Decoder. Encoder models transfer images into **lower dimension latent space**, and in the decoder part we used these latent features to **reconstruct** the same image. We will use **Mean Squared Error(MSE)** to find the **loss** between input image and reconstructed image.

Our hypothesis is that If we are able to reconstruct the input image using lower dimensional features, we can say that we effectively compress the image information in latent space. After training AutoEncoder we will pass every training image into the encoder part of the model and store generated lower dimensional embedding into datastore. At run time we can pass the test query image to encoder and generate the query embedding, then we can compare the query embedding with all the training image embedding stored in the data store, and then return top k results.

Encoders follow the typical architecture of a convolutional network. It consists of the repeated application of 3x3 convolutions (*unpadded convolutions*), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with *stride 2 for downsampling*. We use 16,32, and 64 filters. Every step in the decoder path consists of an upsampling of the feature map followed by a 3x3 convolution (*up-convolution*). We will use 64, 32, and 16 filters in the decoder layer.

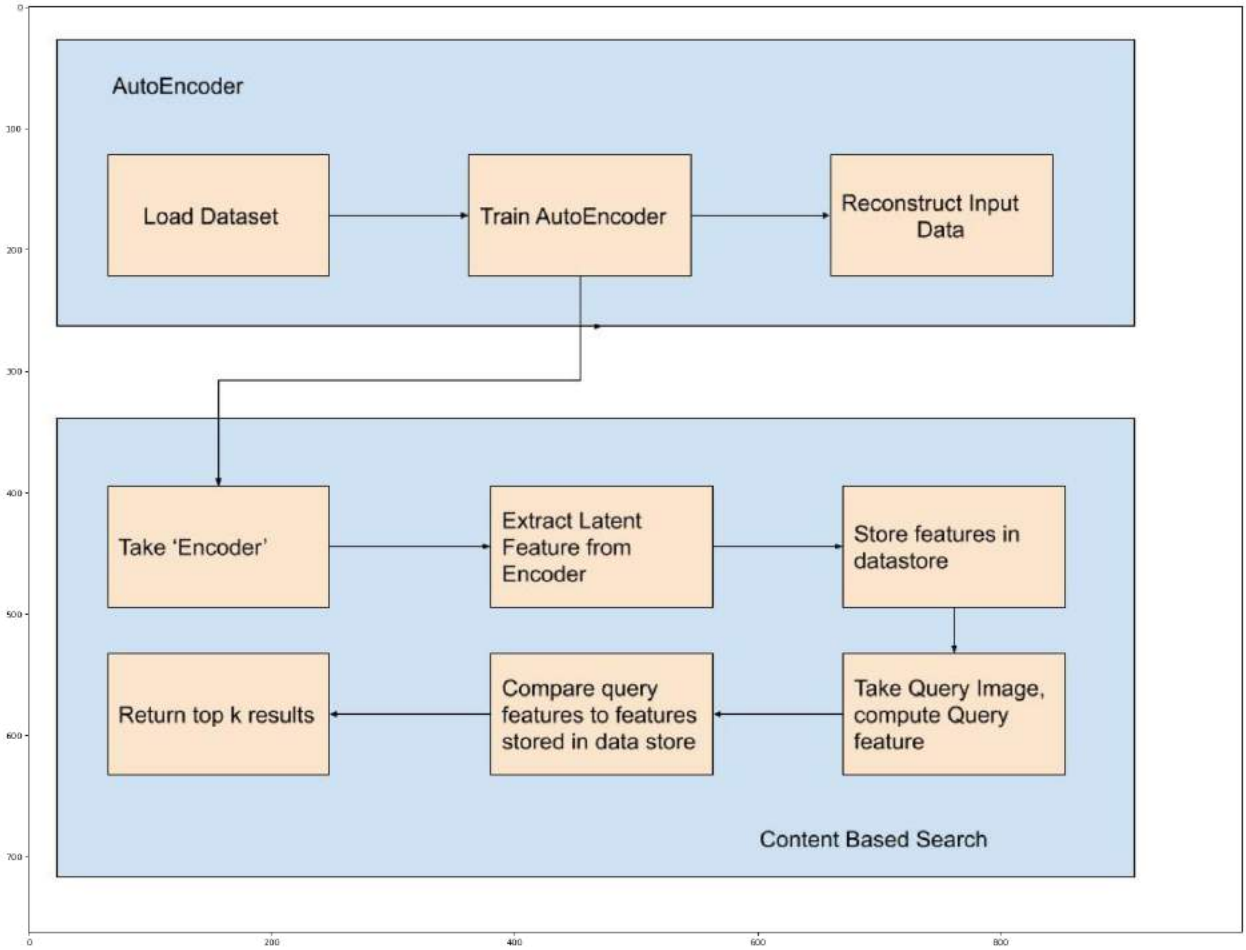


Figure 4 Workflow

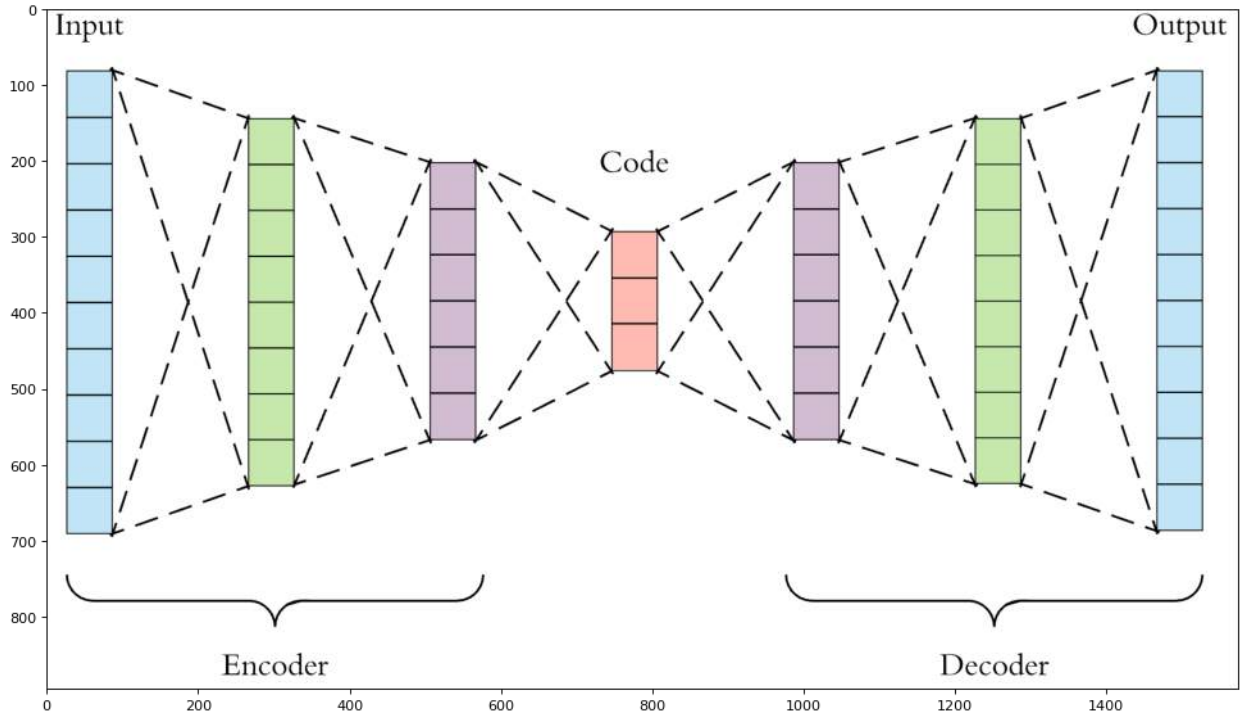


Figure 5 Architecture

5.2 Text model using word embedding algorithms

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. Hence, we will be exploring a few popular techniques to come up with the best word embeddings for our dataset, allowing us to group similar titles together.
For all the techniques we used cosine similarity as our similarity measure & tested with different thresholds i.e `[0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85]`

First we explored **Inverse Document Frequency** to come up with word embeddings. All non-zero term counts are set to 1 so the tf term in tf-idf is binary(0 or 1) i.e essentially we use `TfidfVectorizer(binary=True)`.IDF answers questions like how important a word 'x' in the entire list of documents and is it a common theme in all the documents.

Next we used **Term-Frequency(TF) & Inverse Document Frequency(IDF)** both by using `TfidfVectorizer(binary=False)`.TF answers questions like - how many times is word 'x' used in that entire document i.e a probability. Mathematically ;

IDF =Log[(# Number of documents) / (Number of documents containing the word)]

TF = (Number of repetitions of word in a document) / (# of words in a document)

We additionally explored **GloVe embeddings** against our dataset. GloVe stands for global vectors for word representation. It is an unsupervised learning algorithm developed by Stanford for generating word embeddings by aggregating global word-word co-occurrence matrices from a corpus. The resulting embeddings show interesting linear substructures of the word in vector space.
By using straightforward GloVe embeddings we weren't able to get good results. Hence we combined all the TF-IDF & GloVe and derived **"IDF-weighted GloVe embeddings"** . Hence we combined TF-IDF weights and the pre-trained word embeddings for each word in the document to generate a more sophisticated document vector as follows:

```
embed+=(glove_vectors[word]*key_idf[word])
```

6 Results & Future Scope

6.1 AutoEncoder Reconstruction after 2 epoch

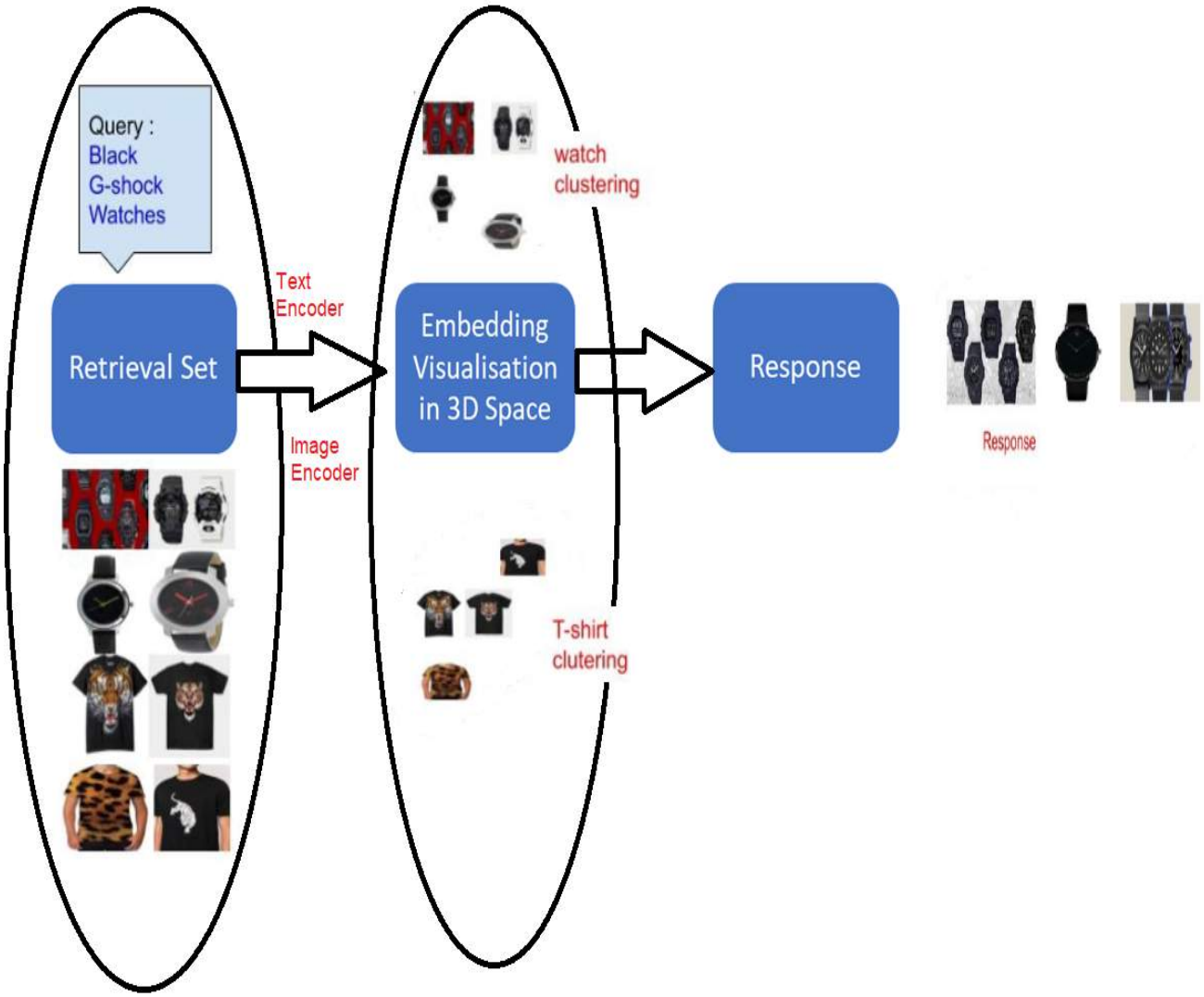
Figure 6

6.2 Testing AutoEncoder and return top 5 result

Figure 7

In the above image, the first image is a query image, and the rest are top 5 similar images returned by our autoencoder model. We are not satisfied with the result. The first 3 images look similar, but the other three images are completely useless and random. Top 3 images have white and pink backgrounds, but actual products are different (first one is a bag, second and third are beauty products), they also have lots of pink and white pixel overlap. In the last three results we have lots of white and light pixels that's why it returned embedding having the smallest distance. The last three images returned because there are lots of white and light pixels in the query image and returned results. This clearly shows that Auto Encoder is not able to understand semantic meaning. If the user is searching for pink bags we would like to return them pink bags and in the worst case any type of bag if we don't have pink bags. In next iteration, We would like to further investigate how we could effectively capture semantic

meaning based on query.



6.3 Comparison of F1 score for word embedding algorithms

Cosine Similarity Threshold/Methodology	IDF	TF-IDF	GloVE	GloVE+IDF Weighted
0.5	0.659	0.637	0.032	0.044
0.55	**0.661**	**0.648**	0.057	0.079
0.6	0.650	0.645	0.095	0.129
0.65	0.633	0.634	0.296	0.366
0.7	0.613	0.617	0.389	0.453
0.75	0.590	0.595	0.473	0.516
0.8	0.567	0.570	**0.518**	**0.536**
0.85	0.542	0.545	0.513	0.518

Table 1

From Table 1 we observe that using only IDF works the best for our dataset with an F1-score of 0.661 for a cosine similarity threshold of 0.55. Hence we will be using the TF-IDF vectorizer with only IDF as our word embedding model. We did think that IDF weighted GloVe embedding model would give us the best results however our results showed just the opposite. On further analysis we realized that only 45.578% of the unique words in our dataset have glove vectors without translation. Therefore in order to fully reap the benefits of using GloVe vectors we will need to translate the entire dataset to English. Hence in Phase 2, we plan to redesign our dataset by adding an additional pre-processing step of translation to English using the Google Translate API. Additionally we will be testing with BERT & FastText as well. FastText offers multilingual support & has pre-trained word embeddings. BERT on the other hand gives importance to context and we hope for better results.