A Picture is Worth a Thousand Feature Vectors: An Image Search Engine

A woman walking down a street past a doorway,
a large slice of angel food cake, and a group of traffic lights

December 2018

Abstract

For the final exam of CS5785 Applied Machine Learning at Cornell Tech, we were challenged to identify relevant images based on a natural language query. Our final model utilized image features extract by ResNet, Singular Value Decomposition (SVD), Term Frequency - Inverse Document Frequency (TF-IDF), Partial Least Squares Regression (PLSR), and Cosine similarity. This model scored 31.1% correctly on Kaggle.

1 Introduction

Google has indexed over 10 billion images on the internet as of 2010 [1]. As the number of images indexed continues grow rapidly, companies like Google are extremely interested in ways to improve the accuracy and efficiency of their image retrieval algorithms. In this paper, we explore various approaches toward retrieving relevant images given a natural language query. For the training and testing image pools, we used features extracted by ResNet, a state-of-the-art Deep-learned CNN that achieved significant results in image classification during the ImageNet competition in 2015 [2].

2 Competition Description

The competition focused on searching for images given a natural language query [3]. Students were provided with 10,000 training samples, containing a JPEG image, a list of tags, feature vectors extracted from the pool5 and fc1000 layers of ResNet, and a five-sentence description. For testing, our model matched a single five-sentence description against a pool of 2,000 candidate samples from the test set. Each candidate sample had a JPEG image, a list of tags, and ResNet feature vectors. For scoring, our model output 20 likely images for each candidate sample. There is one true image for each candidate sample. The system evaluated our rankings using MAP@20. For more details, reference the Kaggle competition overview page [3].

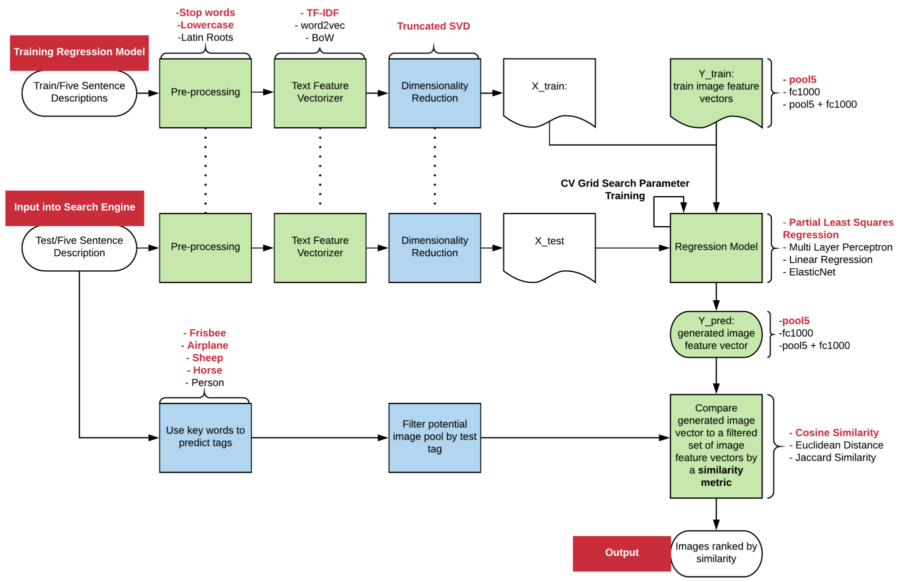


Figure 1: General model framework. Notes on various sub-problems describe the various methods attempted. The methods bolded in red reflect the method used in our final model.

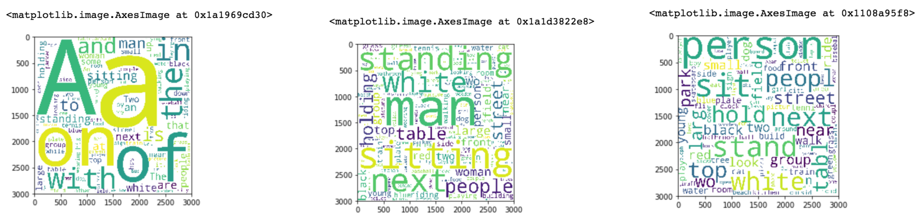
3 Model Architecture

While we tried several methods, our models followed a general framework, described in Figure 1. We divided the overall problem into several sub-problems:

- Pre-processing the sentence descriptions
- Representing the processed sentences as a word feature vector
- Mapping a word feature vector into the ResNet image feature vector space (pool5/fc1000)
- Identifying the correct image feature similarity metric that corresponded to content similarity

3.1 Pre-processing the sentence descriptions

We extracted each five sentence description into a single string and performed some basic pre-processing steps. We lower-cased all words, removed anything that was not a letter or number, and removed basic stop words provided by the nltk library. While processing, we chose to keep word duplicates in order to preserve frequency information for our word feature vector.



```
<matplotlib.image.AxesImage at 0x1a1969cd30>
<matplotlib.image.AxesImage at 0x1a1d3822e8>
<matplotlib.image.AxesImage at 0x1108a95f8>
```

(a) No pre-processing (b) Stop words removed (c) Latin Roots

Figure 2: Word clouds representing word frequency at various stages of processing.

Additionally, we investigated converting each word into its Latin root (i.e. the words “clothing”, “clothes”, “cloth” were all reduced to the word “cloth”). This allowed us to reduce our number of unique words from 9725 to 7430, but ultimately, this reduced our model accuracy, signalling that there was some loss of information through conversion. Figure 2 visualizes the word cloud of most frequent words at various stages of processing.

3.2 Representing the processed sentences as a word feature vector

While we began with a simple bag of words feature vector, we found that representing our sentences with term frequency - inverse document frequency (TF-IDF) was much more effective. TF-IDF is a numerical statistical method used to reflect the importance of a word in a document. The formulas for TF-IDF are as follows[4]:

$$\text{Term Frequency}(t,d) = \frac{\text{\# of times term t appears in a document}}{\text{Total \# of terms in the document}}$$

$$\text{Inverse Document Frequency}(t,D) = log_e\left(\frac{\text{Total \# of documents}}{\text{Number of \# with term t in it}}\right)$$

$$\text{TF-IDF}(t,d,D) = \text{Term Frequency}(t,d) * \text{Inverse Document Frequency}(t,D)$$

Term frequency increases the importance of a word if it is used often in a document and Inverse Document Frequency is used to diminish the importance of words that are used extremely frequently throughout all documents. This is useful in natural language processing for distinguishing useful signals from noise.

Additionally, we also tried word2vec from the gensim library trained with model weights trained from Google News. Ultimately, TF-IDF gave us the best performance.

3.2.1 Dimensionality Reduction

We decided to reduce our TF-IDF feature vector as it would reduce the time it would take to run a regression model on it without any loss in performance. We found the proper number of dimensions by making sure we select enough dimensions to reach our target variance.

Then, we used TruncatedSVD to to reduce our components in TF-IDF from 9717 to 3000. Running the model with more dimensions than 3000 gave a very negligible difference in our results.

3.3 Mapping a word feature vector into the ResNet image feature vector space (pool5/fc1000)

This sub-problem was arguably the most complex and open-ended part of the project. At its heart, this is a regression problem.

Deciding on using pool5 vs. fc1000: we ran our models on both and we found that pool5 works quite well with a TF-IDF word feature vector. This is probably because pool5 comes before the fc1000 layer in the ResNet and contains more information about the images than fc1000. We know that the dimensions of fc1000 are some sort of class labels, so it should map better to only nouns in our description sentences. However, we hypothesized that pool5 captures more semantic value of the image. So, although it might be harder to understand what exactly the 2048 dimensions are in pool5, it gave us better results.

3.3.1 Choosing Our Regression Model

We first tried Multi-Layer Perceptron (MLP) model and obtained decent performance. After doing parameter tuning with CVGrid Search, we got up to 19% on Kaggle.

We obtained better performance with a Partial Least Squares Regression (PLSR). When reviewing the literature, we found that PLSR is a method to model an output when there are a large number of input variables, and those inputs are highly correlated [5].

Similarly, we ran CVGrid Search to determine the optimal number of components to use with our PLSR model.

3.4 Identifying the correct image feature similarity metric that corresponded to content similarity

We began with a simple Euclidean distance measure, but soon found that Cosine Similarity offered better performance. Cosine similarity measures the angle between two vectors, but it not concerned about the magnitude of the individual vectors.

$$\text{Cosine Similarity}(X, Y) = \frac{X \cdot Y}{||X|| \cdot ||Y||}$$

This is useful when comparing two pool5 vectors, as the columns are weights related to correlation of a specific concept. If one image has a very strong weight on a "zebra" concept and another image has a moderate weight on "zebra", Euclidean distance may report a bad fit but cosine similarity will report a good fit since the vectors are pointing in the same direction.

3.4.1 Predicting the tags for given test descriptions

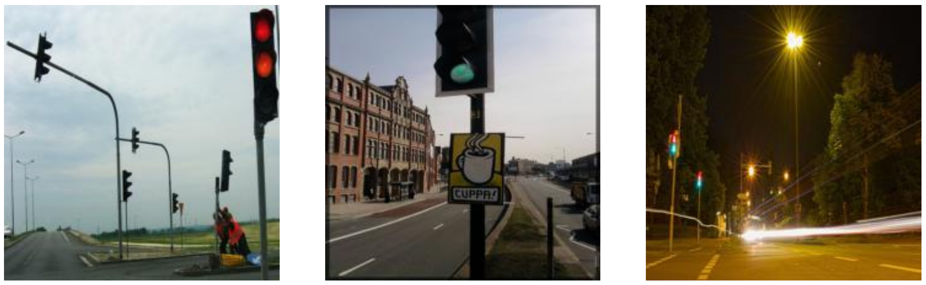
We started with first determining the total number of unique tags and unique categories which we found to be 80 tags with 12 categories which were the same for both train and test data. We had two different approaches to utilizing tags. First, we determined the support for different tags and found more than 40 percent of the images to be labeled with person. We used this information to create a model for filtering the pool5 values used for predicted the nearest neighbor for each test image. By creating two different pools for images with the tag person vs the images without, we were able to increase our accuracy score by 1.2 percent.
Using this sanity check, we created a model to predict the tags for each test description text. Our Y train model was the feature vector representing the 80 different tags with the ones present flagged with bits flipped for the tags present in each description. Our X train was the original feature vector we created for train descriptions with only nouns and stop words removed. We predicted the tags for the test data using an SVM kernel set to 'linear'. The binary string were not accurate, so we decided to drop this approach.

Ultimately, we found a subset of highly specific tags that the images were reliably tagged with and would not be mistaken by other common English words (e.g. toothbrush, sheep, cake, etc.). We used this filter the pool of possible neighbors, which improved performance over our original model slightly.

4 Results

Table 1 summarizes the results of our various models. Our most successful model produced a score of 31.1%. It used TF-IDF, tags for filtering, SVD, pool5, PLSR, and Cosine similarity.

Word Feature Vector	**Using Tags**	**Word FV Dim. Reduc.**	**Image FV**	**Regressor**	**Similarity Measure**	**Score**
	Filtering	SVD(n=3000)	pool5	PLSR	Cosine	0.31108
TD-IDF	No	SVD(n=3000)	pool5	PLSR	Cosine	0.30402
TD-IDF	No	SVD(n=2581)	pool5	PLSR	Cosine	0.30140
Bag of Words	Merged	SVD (n=3000)	pool5	PLSR	Cosine	0.29718
TD-IDF	No	SVD (n=3000)	pool5, fc1000 ensembled	PLSR	Cosine	0.25161
TD-IDF	No	SVD (n=3000)	pool5, fc1000 concat.	PLSR	Cosine	0.25903
TD-IDF	No	No	pool5	MLP	Cosine	0.22985
TD-IDF	No	No	FC1000	MLP (tuned)	Euclidean	0.19742
TD-IDF	No	No	FC1000	MLP	Euclidean	0.14423
Bag of Words	No	No	FC1000	MLP	Euclidean	0.09510



(a) 838.jpg (b) 634.jpg (c) 445.jpg

Figure 3: What did we classify well

Description 002.txt: A group of traffic lights sitting above an intersection. The sign shining down over the street lights A picture of a stoplight from a window. The sun surrounded by an airplane and traffic lights Several street lights and an airplane flying overhead.

4.1 Next Steps:

- We need a better way to predict our tags based on our test descriptions. We were thinking of running a SVM linear kernel regression to predict tags and this should have helped us better classify our images with the help of tags.
- As always, the best way to create a ensemble method based on different models for different sources of data is going to work quite well. We experimented with different methods of mixing and matching but of course, more time is needed to create a better mixture of models.
- One of the main frustrations we faced was how slowly our laptops were running our regressions. Tensorflow would be ideal in the future to speed up our processing power.
- Given more time, we would have done better pre-processing on our sentences, like extracting n-grams and nouns, adjectives and verbs to capture semantic meaning in a text description.

5 Open Source Libraries Utilized

pandas - reading and managing structured data numpy - manipulation of arrays and matrices nltk - natural language processing tools for tokenizing and stopword removal sckit-learn - common machine learning models gensim - word2vec

6 Acknowledgements

The authors would like to thank Professor Serge Belongie, Ryan Benmalek, and Valts Blukis for their excellent instruction and advice.

7 References

[1] https://googleblog.blogspot.com/2010/07/ooh-ahh-google-images-presents-nicer.html
[2] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[J]. arXiv preprint arXiv:1512.03385, 2015. https://arxiv.org/abs/1512.03385
[3] https://www.kaggle.com/c/cs5785-fall18-final
[4] http://www.tfidf.com/
[5] https://www.mathworks.com/help/stats/examples/partial-least-squares-regression-and-principal-components-regression.html