Building a large-scale image search engine with Deep Learning

Anonymous writer
Cornell Tech
aw@cornell.edu

Anonymous writer
Cornell Tech
aw@cornell.edu

Abstract

This papers addresses the problem of retrieving relevant images given a natural language query. We used image features extracted from a state of the art deep-learned convolutional neural network (ResNet) and combined them with natural language processing methods to develop a ranking framework. Specifically, we used a pre-trained word embedding model in order to map the corpus of the image descriptions to a high dimensional vector space. In this paper we present two different methods for leveraging the above framework to rank-order the most relevant images from a large database. The first one is based on regression methods to generate ResNet-like features for each description in order to use different similarity measures for comparison with the database images. The second one is a hybrid approach that leverages the first approach in order to transform the problem to a binary classification setting.

1 Introduction

With the universal popularity of digital devices embedded with cameras and the fast development of Internet technology, billions of people are projected to the Web sharing and browsing photos [6]. It has therefore become vital to be able to retrieve relevant visual documents from a natural language query, giving rise to the field of Image Search, which still attracts a lot of attention mainly due to the emergence of new techniques.

Recently, with the data revolution, research areas such as Deep Learning and NLP have had a remarkable success in the field of Computer Vision. Specifically, the ResNet architecture, which is also used in this paper, has achieved a 3.57 % error on the ImageNet test set.

1.1 Problem

As part of the CS5785 Fall 2019 Final at Cornell Tech, we developed an algorithm in order to build a better search engine using novel statistical learning techniques.

To tackle the problem, we are given $10,000$ JPG images, together with a list of tags describing the objects that appear in the picture, a five sentence natural language description of the image, and features extracted from the ResNet. Specifically, we have access to both the feature vectors of the pool5 and the fc1000 layers of the network.

The evaluation metric is Mean Average Precision at 20 (MAP@20) on the test set. In our case, there is only one true corresponding image. So for a description, suppose your algorithm finds the corresponding image in the list of top 20 images you return, and is ranked i-th. Then MAP@20 gives a score of

$$score = \frac{1}{i} \tag{1}$$

For example, if it's ranked 1st you get a score of 1, if it's ranked 2nd you get a score of 0.5, if it's ranked 3rd you get a score of 0.333, if it's ranked 20th you get a score of 0.05, etc. If the corresponding image isn't in your list of top 20 images you get a score of 0. The overall score is calculated as the average score on the test set.

2 Model Framework & Architecture

Two approaches are presented in this paper. Before diving deep into each one, it's worth describing some aspects that are shared in both. Firstly, to be able to utilize the natural language descriptions, we need a way to parse them into something useful. For this we use the word2vec embedding approach [3] to transform each word of the description into a high dimensional vector space. Utilizing a pre-trained implementation of this approach, we obtain a 300-dimensional vector representation of each description.

Having done this, we can now formulate the following supervised learning problem. Given an image, we can use the word2vec embedding in order to predict the ResNet features associated to each image. This is in essence a multiple target regression problem where we are using 300 variables to predict 1000 response variables (from the fc1000 layer).

2.1 Pre-Processing

An important aspect of both methods is the pre-processing applied to the data. It's therefore worth mentioning a few words before going into each method in detail. Firstly, standard pre-processing has been applied to the natural language descriptions. Lower-casing, removing stopwords and lemmatizing ensures the word2vec embedding is as representative of the description as possible and doesn't take into account any noise associated to stopwords, punctuation etc.

Another important aspect of the pre-processing is the dimensionality reduction of the ResNet features. Originally, the features extracted from the ResNet are 1000 dimensional vectors. To facilitate regression and boost its performance, we note that 100 dimensions are enough to capture around 90% of the variance captured by these vectors. We therefore used PCA to go from 1000 to 100 dimensional ResNet features.

2.2 First Method: From Regression to MAP score

The first method is basically what has been described above. After training a model that can transform an embedded description to a ResNet-like feature vector, we can apply it to the test set descriptions and obtain feature vectors. The problem then boils down to finding a way to measure the similarity between the output of the model and the actual fc1000 features of the test set images. We can then calculate a ranking and hence an MAP depending on the position of the actual image in the ranking. In the experimentation section, we discuss different approaches to calculate this similarity.

2.3 Second Method: From Regression to Classification with Random Sampling

Comparing vectors in high dimensional spaces can be problematic. Recent research results show that in high dimensional space, the concept of proximity, distance or nearest neighbor may not even be qualitatively meaningful [1]. In an attempt to address this, we used a developed hybrid method, mixing regression and classification to make our predictions. To begin, we split the 10,000 training samples into a training set of size 8000 and a development set of size 2000. We then trained a regression model on the 8000 word2vec training samples to mimic each's associated ResNet feature, as above. The regression model we trained was a vanilla neural net with 600 nodes and relu treatment and 50 epochs. We chose a wide, shallow neural net in order to keep a low bias between the predicted ResNet features and real ResNet features. This will be useful to build a discerning classification algorithm.

Next our goal is to build a classification algorithm. This algorithm will classify whether a

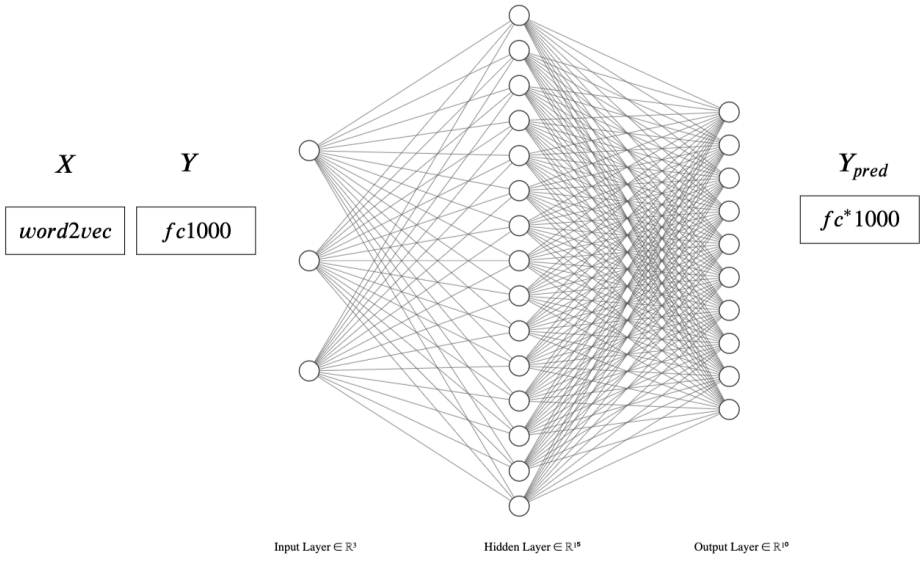

Figure 1: NN architecture for regression. Predicting ResNet features using the word2vec embedding.

predicted ResNet feature set is associated with its real counterpart or not when concatenated together. To build the training set we first add the concatenation of each predicted ResNet feature with its associated real set as a positive sample.

Negative samples will consist of predicted ResNet features concatenated with other real ResNet features from the training set from different images. We began in early trials by picking these samples randomly. However, we found that this led our models to perform poorly. The algorithm was doing well classifying closely associated images, say if our test sample was a picture of someone surfing on a beach, the predicted most likely images were other pictures of beaches, however not the one that it was truly associated with it. In essence, the classifier was good at determining if two images were closely related in subject matter, however bad at determining true images from others that were similar to it. This led us to pursuing a smarter way to pick the negative samples.

We took into account the list of tags available in the dataset for each image. A tag is a short phrase of items or activities appearing in the image. Across the entire dataset there are 99 unique tags. Using these tag features we were able to define a notion of picture similarity. Two pictures are similar if they share many tags. From these tags we built a picture by picture similarity matrix. With the images on both the rows and the columns, for each row each column entry is the number of shared tags those images have. Equipped with this matrix, in order to boost the performance of the classifier between similar images we can chose a certain number of the negative samples for the training. Thus, for the negative entries in our training set we concatenated each predicted ResNet feature vector with a sample of 5 real feature vectors from within the most 500 similar images to that original image, 10 from within the 500-2000 most similar images, and 10 from the 2000 least similar images. We have ensured that the training set includes images that will make the classifier perform better in discerning similar images from each other. For the classification algorithm we chose a neural net with 3 hidden layers, 300, 500, 300 nodes per layer, and relu treatment.

At this stage we have a regression algorithm to transform word2vec features to predicted ResNet features and a classification algorithm that can predict if the concatenation of a predicted ResNet feature set and a real resent feature set are from the same image. Now in order to rank our

images we run the test set through the regression algorithm. After we concatenate each predicted feature set with each real feature set and run it through the classification algorithm to predict the likelihood that the two concatenated sets are from the same image. Our submission is then the top 20 most likely images. [1]

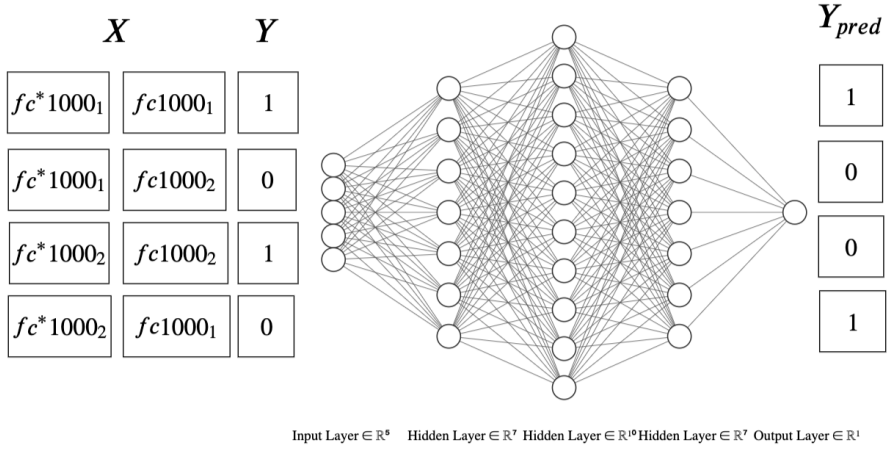

Figure 2: NN architecture for classification. Matching the generated and real ResNet features.

3 Experimentation

Before implementing the final model, we experimented with different approaches on setting up different sections of the architecture. Here, we present a few of them.

3.1 Averaging of descriptions

Instead of embedding each word of the query and averaging them out, we tried summing up each description's words (there are 5 descriptions per query) and then averaging them out. This led to inconsistent results from run to run, so we simply returned to the original approach.

3.2 Utilizing the pooling layer

We also attempted to replace features of the fc1000 layer with the pool5 layer in our approach. While this slightly improved the overall score, we noticed that in significantly increased the mean position of the true image in the sorted list. By investigating this further, we realized that a lot of the images were in the top three, whereas some others were very low in the ranking.

3.3 Similarity measures

When implementing the regression approach (section 2.2), one needs to consider a similarity measure in order to compare the output of the model with the feature descriptions of the test set. We experimented with different measures to figure out which one performs the best.

[1] Note that in both figures shown, the number of nodes is not what is used in the actual model. The purpose is to capture the architecture and the underlying structure of the networks. Refer to sections 2.2 and 2.3 for the exact choices of these parameters.

3.3.1 First Attempt: Euclidean Distance

We naturally started using the euclidean distance between the two feature vectors

$$d(x, y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2} \tag{2}$$

3.3.2 Second Attempt: Manhattan distance

$$d(x, y) = \sum_{i=0}^{n} |x_i - y_i| \tag{3}$$

Research has shown that the behavior of the commonly used L_k norm in high dimensionality is sensitive to the value of k [4]. Thus the Manhattan distance might be more preferable in certain applications. However, in this specific example, the L_1 norm ended up doing worse than the Euclidean distance.

3.3.3 Third Attempt: Cosine Similarity

We finally computed the pairwise cosine similarities between the feature vectors using

$$\cos(x, y) = \frac{x \cdot y}{\|x\|\|y\|} \tag{4}$$

The cosine similarity gave us the best results.

3.4 Negative sampling

Returning to the hybrid regression and classification methods, the most important tuning parameter of this model is the way the negative sampling is performed. By selecting different ways to perform the sampling, we noticed a considerable change in performance. In the next section, we present our findings for different combinations of these parameters. Before doing this, it is important to explain what they are. As mentioned previously, we have constructed a square matrix, that records how many tags different images have in common. This in essence gives us a way to assess the pairwise similarity of images. Now in order to efficiently train a classifier, we need to make sure that our negative samples cover all the different possible scenarios. We therefore divided the negative samples into three categories. A mismatch with a similar image (a lot of tags in common), a mismatch with a neutral image (mean number of similar tags in common) and mismatch of completely unrelated images (no tags in common). This allows us to train a diverse classifier that can detect the subtle differences between features vectors and rank them accordingly. Having this in mind, we have the following tuning parameters.

num_sim: The number of very similar images we add to the dataset.

num_med: The number of relatively similar images we add to the dataset.

num_diff: The number of completely different images we add to the dataset.

sim_offset: How far do we go in the similarity matrix in order to randomly pick similar a num_sim amount of similar images. If this number is low (relative to the number of columns of the matrix), then we're only allowed to pick from the set of very similar images.

med_offset: Same parameter for the num_med images.

diff_offset: Same parameter for the num_diff images. If this number is big, we're making sure that we only pick images that have no tags in common with the original image.

4 Results

We firstly show the loss function of the regression method. Notice here that the regression model remains the same in both approaches so it's important to make sure its performance is satisfactory.

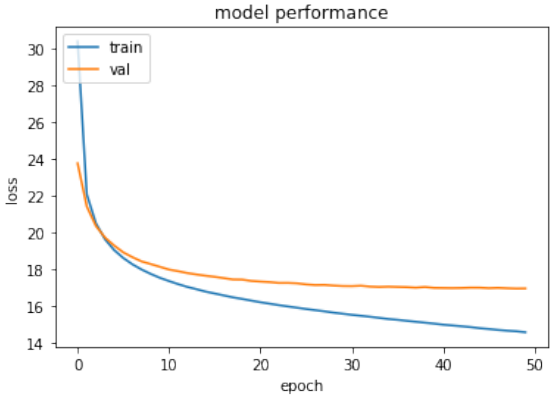

Figure 3: Mean Squared Error of the Neural Network regressor.

We now move to results of each particular method.

4.1 Vanilla Regression: Results as we change the similarity measure

As mentioned previously, the main parameter to experiment with in the regression setting is the similarity measure to compare predicted with feature with the actual images. The table below summarized the performance of the model for three different choices of such measures.

Norm	Map
L_1	0.152
L_2	0.183
Cosine Similarity	0.215

Table 1: Different similarity measures to compare feature vectors.

We observe that cosine similarity performs best. Intuitively, in a highly dimensional space, where points are very far apart from each other, it seems more appropriate to study if two vectors are pointing in the same direction instead of calculating the distance between two points.

4.2 Hybrid Approach: Tuning the negative sampling parameters

We now present MAP score results for different choices of parameters for the negative sampling

num_sim	num_med	num_diff	sim_offset	med_offset	diff_offset	MAP
10	5	5	100	2000	5000	0.242
6	3	3	100	2000	5000	0.222
6	3	3	500	2000	5000	0.237
10	5	5	500	2000	5000	0.253
10	3	3	500	2000	5000	0.232
5	10	10	500	2000	5000	0.265
5	10	10	500	3000	6000	0.261
3	6	6	500	2000	5000	0.238
3	2	2	500	2000	5000	0.201
10	20	20	500	2000	5000	0.269

Table 2: MAP score for different combinations of negative sampling parameters.

Now, equipped with the best combination of parameters, we can inspect the actual ranking and see how would the search engine look like. Firstly, we display an example that got a perfect score, i.e. the true image was at the top of the ranking. Therefore, with an input query of

```
A table topped with trays full of hot dogs.
The table is set up with the food for people to grab what they like.
A table of food that includes hot-dogs and desserts. A table covered in hot-dogs, soc
a table with many different food items and a cake.
```

the model returns

Figure 4: Top 8 images returned by the model. Here the first image corresponds to the real image.

We can see that the model is doing a good job at capturing most of the elements in the description such as food, deserts and sodas. Moving to the most interesting case of an occasion where the score was very low, namely the actual image did not end up high in the ranking, for the following query

```
A tow truck driving past a very old stop sign.
A red stop sign sitting on the side of a road.
A flat bed truck is driving past a stop sign.
A work truck is driving down a dirt road, past a stop sign.
A stop sign at an intersection with a truck going by to the right.
```

the model outputs

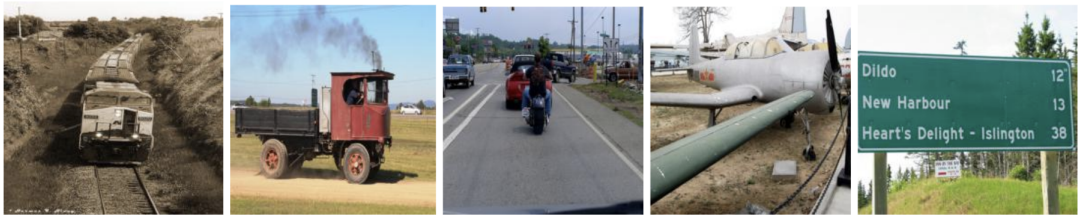

Figure 5: Case where the real image is not included in the top result. Output images are still relevant.

Interestingly enough, the model is doing well at outputting relevant images. A lot of the features from the description, such as trucks dirt road and signs are captured in the output. We can further

observe that the model is having trouble capturing the combined meaning of two words together. Although, the word sign is definitely captured in the output, there are no red signs and the red color is arbitrarily assigned to the trucks. This is not really unexpected since we simply used the mean of each word's embedding to construct the predictor variables. What is more surprising is that although the actual output is quite relevant, the true image is very low in the ranking. Two remarks should be made here. Firstly, the MAP score is not necessarily the most appropriate way to measure the performance of the search engine, since it only takes into account where the true image ended up in the final ranking without measuring how relevant the output images are. Furthermore, since in reality there are no exact matches between images and users' queries, finding a way to capture relevance of the output images would be much more appropriate.

5 Conclusion & further work

Finally, by looking at the distribution of probabilities that the neural network outputs, we observed a very skewed behavior with two modes. Probabilities were either very low or very high. In fact the issue with many deep neural networks is that, although they tend to perform well for prediction, their estimated predicted probabilities produced by the output of a softmax layer can not reliably be used as the true probabilities (as a confidence for each label). One must then employ modern techniques such as confidence calibration can be used to address the problem of predicting probability estimates representative of the true correctness likelihood [2].

In total, despite some of the flaws in our algorithm, we were still quite successful in creating a query system to find relevant images from a natural language search. Throughout development the median position of our true image in our image rankings was 6, and we could see that consistently the most relevant images output where very close to what we wanted. In the future to further improve our model we would work to improve the classifier. We would attempt to use an SVM model instead of a neural network. Furthermore, we would also look into different ways to parse the natural language into a vector space, in order be able to capture the meaning of consecutive words (such as the red sign case presented above).

References

[1] Aggarwal, C.C., Hinneburg, A. and Keim, D.A., 2001, January. On the surprising behavior of distance metrics in high dimensional space. In International conference on database theory (pp. 420-434). Springer, Berlin, Heidelberg.

[2] Guo, C., Pleiss, G., Sun, Y. and Weinberger, K.Q., 2017, August. On calibration of modern neural networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 1321-1330). JMLR. org.

[3] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[4] Sultana, F., Sufian, A. and Dutta, P., 2018, November. Advancements in Image Classification using Convolutional Neural Network. In 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN) (pp. 122-129). IEEE.

[5] Tasche, D., 2013. The art of probability-of-default curve calibration. Journal of Credit Risk, 9(4), pp.63-103.

[6] Zhou, W., Li, H. and Tian, Q., 2017. Recent advance in content-based image retrieval: A literature survey. arXiv preprint arXiv:1706.06064.

[7] Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X., 2019. Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems.