Recognizing Landmarks in Images
https://github.com/emontag/DSIW-Project

Ephraim Montag (em789), Jacob Lee (hl2278) , Hong Noh (hn325)

Abstract

The rapid increase in shared photos has garnered a significant amount of attention to the classic computer vision task of object recognition. In more detail, the task is to recognize specific objects (labels) in images by using computer vision techniques. By using the annotated dataset of landmark images and labels from the Google Landmark Recognition Challenge on Kaggle, we developed a baseline pipeline of various feature extraction methods and the K-Nearest-Neighbors method. Afterward, we used transfer learning to increase the performance. We utilized PyTorch and Tensorflow to develop the model. The effectiveness of the process was compared to other test results on the Kaggle leaderboard for comparison. The experiments aimed to answer the following questions: How can we reliably and quickly extract feature vectors from images? How can we accurately compare the query image with the training images? How can we set up experiments to build a robust pipeline?

1 Introduction

The task of object recognition in images is important in many different areas such as robot positioning and image query engine. With the wide adoption of social media and mobile phones, there is a vast amount of shared images, and the topic of recognizing objects to automatically label images has become more important than ever. Within the task of recognizing objects in images, recognizing landmarks is particularly interesting because travelers around the world love to share their travel photos. An example is labeling an image of the Eiffel Tower. Despite of the vast amount of images, the challenge in the task is the lack of annotated images to perform supervised learning. In this paper, we utilize the annotated dataset from Google Landmark Recognition Challenge and focus on developing a baseline pipeline by using feature extraction and matching methods and KNN. We evaluated our model by performing cross validation and comparing to the test results on the Kaggle leaderboard.

2 Related Work

We explored various approaches to the task of recognizing landmarks in images. There are three predominant approaches. Image retrieval is a method in which a query into a database returns various clusters of potential matching images. Subsequently, a image is compared to the clusters, and the label of the best matching cluster is assigned to the query image. Quack et al. [1] uses computation-heavy direct feature matching. Li et al. [2] determines whether a specific landmark in the query image using iconic scene graphs. Secondly, classification is another approach to the task by viewing each label as its own class. Li et al. [3] applies SVM and image tag information to classify images. Bergamo et al. uses 1-vs-all SVMs for classification. Lastlz pose estimation is an approach to gauge the camera angle and position of images to robustly extract and match features from images. Nister et al. [4] uses vocabulary trees to perform scalable object recognition.

3 Dataset Description

The dataset came from the Google Landmark Recognition Challenge. The training data consists of clusters corresponding to various landmarks based on geolocation and visual similarity. Y.-T. Zheng et al. [5] show how to cluster landmark images using local feature matching. There was no ground truth algorithm to avoid any bias, and human annotators finalize the ground truth correspondences between test images and labels. The test images contain zero to multiple landmarks. There are 235,778 training images, each training image contains only one landmark and each landmark has one unique id. The Exif data associated with each image may contain additional geolocation information, but they are not required to be used.

4 Baseline Model Description

The model used utilized a K-Nearest-Neighbors approach. In this case, K-Nearest-Neighbors plots the training data's feature vectors and computes the k values closest to the feature vector of the validation (or test) image. K nearest neighbors is an approach that allows a baseline to be made with less computational overhead than a convolutional neural network.

4.1 Experimental Setup KNN

The first part of the project involved using a K-Nearest Neighbors approach. The setup for that is below. The first part of the setup involved downloading ”train.csv” and ”test.csv” from the Kaggle competition website. A script was then run (provided on Kaggle) to download the images (links scraped by Google) based on the links in ”train.csv” and ”test.csv.” The training data and test data was kept separate from each other. The images were extremely large and were turned into thumbnail size so as to not take up a significant amount of space (can be up to 330 GB). Also, only about 20% of the images in train.csv were downloaded due to the links not working properly (images were removed). The second part of the setup involved loading and cleaning up the data. To begin, a list of the images was made to know the names of the associated files. Then a dictionary of filenames to landmark ID's was created. The images were then appended to a list and flattened (turn multi dimensional array into a one dimensional array). Any images that were corrupt were removed. The data was then split using cross validation to be able to validate the model. The third part was setting up the K-Nearest-Neighbors. For this, we set k at 3 and k at 5. We then trained the training data and the training labels and predicted the validation images. For the fourth part, we decided to try different preprocessing techniques on the images to train the model. Edge detection was used on the images and feature vectors were derived from there. The second part, was just take the feature vectors on the images themselves without any feature detection. 1% of the data was used for validation.

4.2 Experimental Setup CNN

The second part of the project involved transfer learning. For that we needed to preprocess the images even further. We separated the files by landmark id in separate folders. We also split the data that 10% of it would become the validation set. We realized that most labels have less than twenty images associated with the. Hence we cannot train our own Convolutional Neural Network. We decided to use the Imagenet resnet50 model for the transfer learning. We used an SGD optimizer. That optimizer used a learning rate of 0.001 and momentum (for acceleration of SGD in relevant direction) of 0.9. We experimented with the number of features to look at. The third thing we decided to do is clean up the dataset further. We only processed labels that had over a different numbers of images per label associated with it. We then ran transfer learning on that dataset. This was done using the inception v3 model trained on Imagenet. The fourth thing we did was make a CNN and see what the results would be. We split the training data into a training set and validation set. One hot encoding was done on the labels.

4.3 Results and Analysis KNN

For KNN, we initially split the data into a 60/40 test/train split. However, upon trying to run our algorithm to predict labels on the test data we found that this test dataset was too large. After several trials we decided to use 99% of the data to train the model and use 1% of the data to test it. Owing to the large amount of data involved, we ran the KNN algorithm on images *after* Canny edge detection and subsequent flattening. The results for two values for K are shown below:

K	F1- Score	Gold-Standard
3	0.001	0.216
5	0.003	0.216

In the second test, we once again used 1% of the data to validate, but this time *without* using Canny edge detection.In this case, the results are shown as below:

K	F1- Score	Gold-Standard
3	0.008	0.216
5	0.007	0.216

The third test was performed with 5% of the training data used for validation, again *without* Canny edge detection. The results for this are shown below

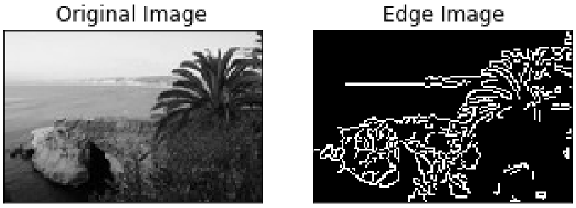
K	F1- Score	Gold-Standard
3	0.007	0.216
5	0.007	0.216

The final test was performed with 10% of the data, again *without* Canny edge detection. The results are shown below:

K	F1- Score	Gold-Standard
3	0.006	0.216
5	0.006	0.216

In all tables above, the 'Gold-Standard' is the current best result as shown on the Kaggle website for the challenge.

Figure 1: Sample image before (left) and after (right) the Canny Edge Detection algorithm has been run.



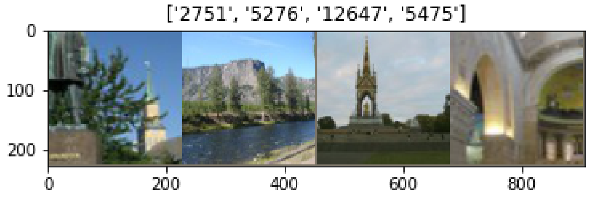
There are several reasons why we could have had such poor results for KNN. First, we were working with compressed image data which may have lost some features that could be discerned. Second, the images had a lot of noise from people and other objects in the image making any feature vectors generated to be full of "bad" data. Third, K Nearest Neighbors relies on clear distinctions between the data so that distances can be computed properly. In this case with images many of the feature vectors would correlate to each other but not actually be similar to each other. furthermore, images of the landmarks were taken at varied angles and distances making K Nearest Neighbors be unable to compute distances as properly as if the data had less noise. Feature detection that extracts he relevant portions of the image are necessary to get it to work. Perhaps cropping that part of the image would increase the detection capability of K nearest Neighbors. As is shown in the image above the edge detection is filled with random noise which skews the model and makes the accuracy lower. perhaps the reason why the feature vector of the whole image was better was because it still kept other features that canny edge detection removes.

As can be seen the accuracy may not be great but the greatest accuracies for the competition top at about 20% which is not very high.

4.4 CNN Results and Analysis

We trained the transfer learning model that was implemented in PyTorch over 5 epochs, with 10% of the training data used for cross validation. This yielded an accuracy result of .0096 accuracy with the numbers of features set to ten. At five features the accuracy was reduced to .007. As we observed low accuracy scores with the model in PyTorch, we explored different frameworks to implement transfer learning. Subsequently, we noticed that the given dataset was particularly unsuitable as most of its classes contained insufficient samples to train the model. With such a limited dataset, we had to preprocess such that each class had enough data for retraining the inception model. In order to retrain, we only selected the labels that had more than twenty images (56 labels out of over 14,000). We split the filtered data into training, validation, and test sets. After running more than 1000 steps, we produced the accuracy score of 0.894 on the test set of 114 images with 56 labels. Although we could not utilize the entire training data, we believe that we can scale this solution to bigger datasets.

Figure 2: Sample images with labels.



The primary challenge in performing transfer learning was the dataset. The low accuracy occurred most likely because there are insufficient data for individual labels to effectively train a CNN model. The frameworks we used required at least 20 images per label for retraining. Our dataset mostly only had labels with fewer than five associated images, making the CNN approach somewhat ineffective and resulting in an underfit model.

Overall, the results on the CNN are not anywhere what we expected using the full dataset. However, we were able to determine that the accuracy would significantly increase if we had a greater number of images per label. If we were able to train a CNN with over the million files that should have been able to be downloaded validation scores my have been better.

5 Conclusion

Transfer learning (CNN) performed marginally better than KNN. This is seemingly not so significant. As discussed above, CNN's require many images per label which we lacked resulting in the above results. We can possibly conclude that with a cleaner dataset and more images CNN's would perform better than KNN did.

6 Citations

[1] T. Quack, B. Leibe, L. Van Gool, World-Scale Mining of Objects and Events from Community Photo Collections, in: Conference on Image and Video Retrieval, 2008, pp. 4756. [2] X. Li, C. Wu, C. Zach, S. Lazebnik, J.-M. Frahm, Modeling and Recognition of Landmark Image Collections Using Iconic Scene Graphs, in: European Conference on Computer Vision, 2008, pp. 427440. [3] Y. Li, D. J. Crandall, D. P. Huttenlocher, Landmark classification in large-scale image collections, in: International Conference on Computer Vision, 2009, pp. 19571964. [4] D. Nister, H. Stewenius, Scalable recognition with a vocabu- lary tree, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2006, pp. 21612168. [5] Y.-T. Zheng, M. Zhao, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher T.-S. Chua, and H. Neven, Tour the World: Building a Web-Scale Landmark Recognition Engine, Proc. CVPR09 [6] H. Noh, A. Araujo, J. Sim, T. Weyand, B. Han, ”Large-Scale Image Retrieval with Attentive Deep Local Features”, Proc. ICCV'17