7/25/2016

PROJECT REPORT

PROJECT 3: Image Based Search Engine

Avijeet Mishra
UNIVERSITY AT BUFFALO
AVIJEETM@BUFFALO.EDU
50169242

INTRODUCTION

This report is based on the explanation of an image search engine and its analysis using real world photos from North Campus of University at Buffalo. Initially 25 images of resolution downgraded to 1280X720 in the dataset is indexed using the code index.py. Indexing is the process of quantifying our dataset by using an image descriptor to extract features from each image and storing the resulting features for later use, which in our case is performing a search. Index.cpickle is used to store the index which would be later used for performing the search using 5 query images of resolution 1280X720.

An image descriptor defines how we are quantifying an image, hence extracting features from an image is called describing an image. The output of an image descriptor is a feature vector, an abstraction of the image itself. Basically it is a list of numbers used to represent an image. The image descriptor used here is a 3 D color histogram in the RGB color space with 8 bins per red, green, and blue channel. When computing a 3D histogram with 8 bins, OpenCV will store the feature vector as an (8, 8, 8) array. It is flatten and reshaped to (512,).

Once it's flattened, feature vectors could be easily compared together for similarity. It's important that we normalize the histogram in terms of pixel counts. If we used the raw (integer) pixel counts of an image, then shrunk it by some percentage and described it again, we would have two different feature vectors for identical images which should be avoided. Scale invariance is obtained by converting the raw integer pixel counts into real-valued percentages. Two feature vectors can be compared using a distance metric. A distance metric is used to determine how "similar" two images are by examining the distance between the two feature vectors.

In the case of an image search engine, a query image is provided to search_external.py and ask it to rank the images in our index based on how relevant they are to our query. Here is the part where the actual searching takes place. Image filenames and corresponding features in the index are looped upon. Then chi-squared distance is computed to compare the color histograms. The computed distance is then stored in the results dictionary, indicating how similar the two images are to each other which is sorted in terms of relevancy (the smaller the chi-squared distance, the relevant/similar) and returned. Images will be considered identical if their feature vectors have a chi-squared distance of zero. The larger the distance gets, the less similar they are.

Finally, path of the top 10 images are fetched and they are displayed to user in two montages, top 5 in 1^{st} and 6-10 in 2^{nd}.

Source: http://www.pyimagesearch.com/2014/01/27/hobbits-and-histograms-a-how-to-guide-to-building-your-first-image-search-engine-in-python/

ANALYSIS

There are 5 test cases comprising of total 25 images used for indexing and 5 query images. For analytical purpose, as we have database of 5 relevant images per query and two scenarios considered which are Top 5 and Top7.

1st Test case: Bridge between Norton and O'Brian Hall

Query Image

Output Generated

Analysis

Top 5: Only 3 relevant results were returned, so accuracy is 3/5 = 0.6

Top 7: All 5 relevant results were returned, so accuracy is 5/7 = 0.71

2 Images that were not returned in Top 5 is because of the reason that both picture of Clemens hall and the bridge is quite similar because of the sky, windows and color of building in the picture.

2nd Test case: Clemens Hall

Query Image

Output Generated

Analysis

Top 5: All 5 relevant results were returned, so accuracy is 5/5 = 1

Top 7: All 5 relevant results were returned, so accuracy is 5/7 = 0.71

All 5 relevant images are returned in Top 5 and it is because of the reason that all the images of Clemens hall covers almost 90% of the picture is covered by hall which is helping the code to find the exact match.

3rd Test case: The Commons

Query Image

Output Generated

Analysis

Top 5: Only 4 relevant results were returned, so accuracy is 4/5 = 0.8

Top 7: Only 4 relevant results were returned, so accuracy is 4/7 = 0.57

1 Image that was not returned in Top 5 or Top 7 even in Top 10 is because of the reason that the particular 5th picture of The Commons was taken with different angle and different exposure of light so this could be considered as human error rather than code error.

4th Test case: Baird Point

Query Image

Output Generated

Analysis

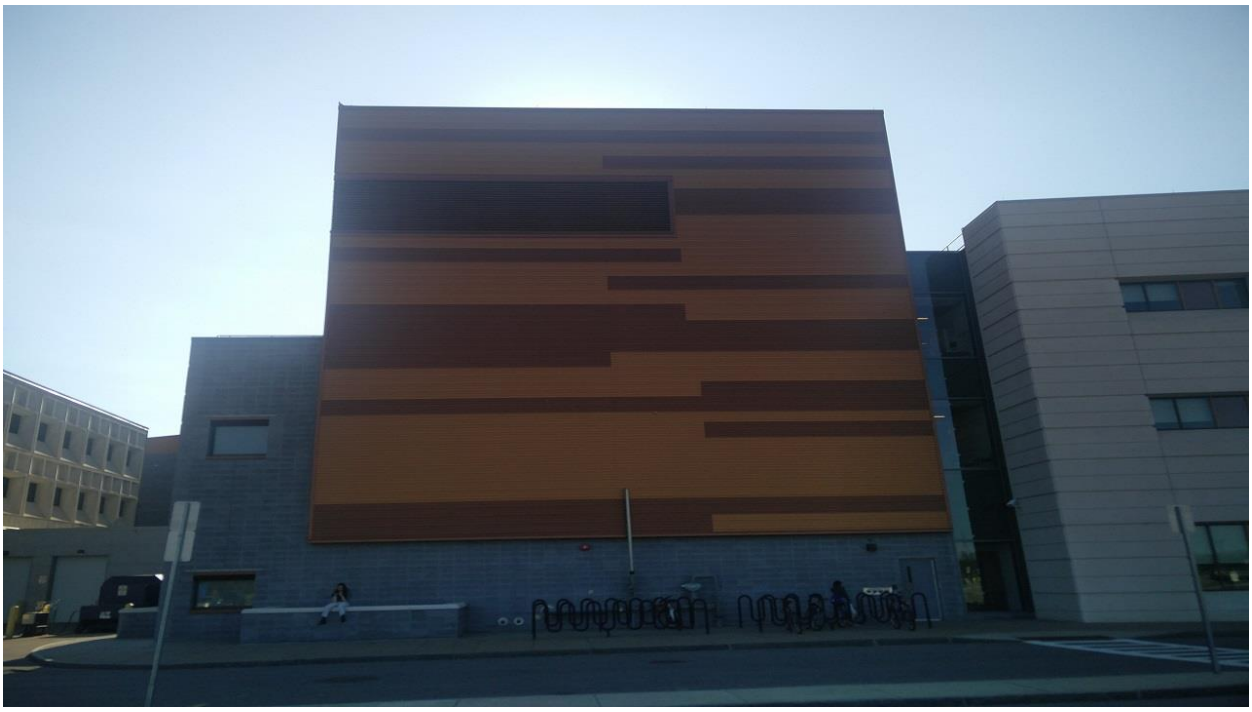
Top 5: All 5 relevant results were returned, so accuracy is 5/5 = 1

Top 7: All 5 relevant results were returned, so accuracy is 5/7 = 0.71

All 5 relevant images are returned in Top 5 and it is because of the reason that all the images of Baird point has maximum area covered with sky and grass, and since camera is very far from the monument and it Is a unique one in all the images, code is able to find the exact match.

5th Test case: Davis Hall

Query Image



Output Generated

Analysis

Top 5: All 5 relevant results were returned, so accuracy is 5/5 = 1

Top 7: All 5 relevant results were returned, so accuracy is 5/7 = 0.71

All 5 relevant images are returned in Top 5 and it is because of the reason that the texture and color of Davis Hall is very unique and code is able to find the exact match.

Analysis of tool taking into consideration all the 5 test cases

Top 5: 22 relevant results were returned, so accuracy is 22/25 = 0.88. It has a very high accuracy of 88%. It could have been more accurate if images would not be similar to each other which is basically because of the sky, grass and similar color of the building.

Top 7: 24 relevant results were returned, so accuracy is 24/35 = 0.69. In Top 7, maximum accuracy that could be achieved is 25/35 = 0.71 as we have only 5 relevant images per query. So taking this into consideration, 0.69 is very close to 0.71 which is in fact is 97% accurate. If one more image has been returned which wasn't fetched because of human error it would be in fact 100% accurate.

CONCLUSION

It could be concluded that this image search engine is very strong considering the results fetched in the experiment conducted. One of the key observation in this experiment was the 2nd test case of "Clemens Hall". The score of Top images were 0.166, 0.369, 0.384 and 0.595 which is quite staggering and very impressive as well. It was the best top scores observed among the other images. This is because of the fact that maximum area of picture was covered with building with similar color and very less of sky.

One of the major takeaway from this experiment is that redundant pixels like sky, grass, road, same color of building, etc. could sometimes create an illusion for the code to detect two images as similar even if they are of different places. In some of the cases, occlusion also caused the score of a very relevant image to be generated very less. There are very good techniques available like SIFT for describing the images so instead of RGB descriptor, SIFT could be used to improve the accuracy of the code.

1 case in which the image of "The Commons" was not detected in even top 10 images because the image was taken in different angle and different exposure of light which eventually changed the color of the resultant image and since color is the major factor in detecting the image, if different colored images are used in test case this tool would fail. It could be considered as human error but observing visually, humans can clearly identify that it is similar image but his tool can't. It could also be because of very extreme angle difference and lot of difference in shadows as seen below in which left one is query image and right one is undetected image.

One thing that could have been improved in the technique used, is tf-idf in indexing as basic indexing is used in this code which is very good for the small dataset but could be cumbersome and, time and space consuming for bigger datasets. Another thing which could be done to improve the performance of this

image search engine is by providing annotations or relevant file name to the images and then taking it into consideration which could help in increasing the score of the relevant images and improve the accuracy. For implementing this, we would have to write some extra code which would increase the running time of this program but would definitely improve the accuracy too.

The code is of utmost quality. It is very well commented with detailed comments for almost each line of code. The code is very well structured which is developed using object oriented concepts. This is one of the major reason of quick response time of this image search engine. It could be further optimized by removing the check of j in lines 59-64 shown below.

```
58          # check to see if the first montage should be used
59          if j < 5:
60                  montageA[j * 720:(j + 1) * 720, :] = result
61
62          # otherwise, the second montage should be used
63          else:
64                  montageB[(j - 5) * 720:((j - 5) + 1) * 720, :] = result
65
```

We can run too similar loops from j (0 to 4) and j (5 to 9) to further reduce the running time since the check for j is removed.