CSC420

Alex K. Chang
1000064681

Project Report

2.1 Introduction

My project is based on Nister and Stewenius's Scalable recognition using a vocabulary tree to find and classify dvd covers. The goal of this task is to be able to recognize DVD covers in the real world and match it with the corresponding DVD cover that is present in our database. The dataset I used in my database is the Stanford's mobile visual search dataset [4], which contains 100 dvd covers used for training, and 4 separate folders with 100 images taken from different cameras with different positions and viewpoints in each of the pictures. I use the images from these 4 folders for testing. Using the technique outlined in the paper [2], we will attempt to classify images using "visual words". The visual words are the image features of an image. We will use the database image features as our training dataset and by using k-means clustering on those features to create the vocabulary tree. Then we use hierarchical scoring to find the top 10 matches between the test image and the database images, and ransac to find the best image from the top 10 matches.

2.2 Methods

There are a few steps that I have used in my algorithm for DVD recognition (e.g. Vocabulary Tree creation, hierarchical scoring, RANSAC)

2.2.1 Vocabulary Tree

One of the main components to this project is building the vocabulary tree and using it to score images. This concept is outlined in the paper by Nister and Stewenius [2]. Using the SIFT descriptors [1] obtained from each of the database images, we concatenated them into a single list of all descriptors of all the database images. Then I ran hierarchical k-means clustering on the descriptors with K = 9, L = 9. K is the branching factor and L is the level of the vocabulary tree, excluding the root level. I used the API from VLfeat, vl_hikmeans [3], to implement the vocabulary tree.

2.2.2 Hierarchical Scoring

The scoring is one of the main parts of the algorithm, it allows a Query image DVD to be propagated down the Vocabulary Tree and matched with the database image that has the most similar path collection of features with the query image. I decided to implement the same hierarchical scoring as outlined in [2]. For each image in our database, we define a n dimensional vector d, where n is the number of nodes in our vocabulary tree excluding the root node. For each node i in the vocabulary tree (excluding the root node), we will mark down the number of descriptor vectors (in this implementation case, it is referring to the 128-length descriptor vectors of each feature from a database image extracted using SIFT) that passes through node i as ni. We will also assign a weight wi of a node i in our vocabulary tree. Thus for each node (excluding the root node) in our vocabulary tree, we will have

$$d_i = n_i * w_i \tag{1}$$

In our database dimensional vector d for each image in our database.
Similarly, we will do this exact same definition for the query image. We define a m dimensional vector q, where m is the number of nodes in our vocabulary tree excluding the root node. Similar to equation (1), we will have a m dimensional vector q such that

$$q_i = m_i * w_i \tag{2}$$

Where mi is also the number of descriptor vectors that passes through node i, and wi is the weight of a node i in our vocabulary tree.

2.2.3 Weights:

The weights in the hierarchical scoring for each node i is

$$w_i = \ln \frac{N}{N_i} \tag{3}$$

where N is the total size of the library of DVD covers, and N_i is the number of DVD covers with at least one descriptor passing through node i. A consequence of this is that this weighting function will downweight the path scores of earlier nodes in the vocabulary tree as the N_i tends to be higher in those nodes compared to the leaf nodes. Ultimately, this results in a TD-IDF scheme.

2.2.4 Scoring

The scoring is very straightforward, since in my implementation, the q and d vectors are precomputed before the scoring. During the scoring, we use the L_1 norm (4) outlined in [2]

$$||q - d|| = 2 + \sum_i (|q_i - d_i| - |q_i| - |d_i|) \tag{4}$$

to obtain the score of a query vector and a database vector. We will then take the top 10 minimum difference to use for the next step

2.2.5 Precomputing d-vector and weights

In my algorithm, I do the precomputation of each of the d and q vectors for a vocabulary tree, before the hierarchical scoring. This is first done by a top down approach where we send all database images down the vocabulary tree to compute the d_i . During this process, we also save the w_i for use when we compute the q_i

```
Function build_dbvectors

  N ← vector of size(all nodes) -1
  For each db image
  n ← vector of size(all nodes) -1
    For each db descriptor
         n_i = n_i + 1  for each node i the descriptor passes on way down to leaf node
         N_i = N_i + 1  for each node if its the db image's first time passing through
.                       this node
    End
  End

w_i = log(N/N_i)
  For each db image
    For i = 1 to size(all nodes)
      n_i = n_i * w_i
    End
  End

Function End
```

We keep the precomputed d_i and q_i for use in our hierarchical scoring function

2.2.6 Homography estimation using RANSAC

The last step of the algorithm is to to use RANSAC on the top 10 matches, to do a homography estimation for each of the images in our top 10. I set trials to 1000 iterations, with k = 100 when running the ransac algorithm. For each iteration, the algorithm will pick out 4 matches and using that to compute a homography matrix, then it will see how many inliers show up for the matrix. The image that returned the highest inlier for any given trial homography matrix was likely our match with the query image. Thus my algorithm will keep the image with the maximum inliers and display that as the result.

2.3 Main Challenges

One of the biggest challenges about this project to me was implementing efficiently the algorithm described from [2]. Space efficiency and memory was especially challenging to me since I was working remotely through ssh most of the time. As a result, the memory allocated for programs are limited for ssh use to the client. Concept wise, the algorithm itself did take a while to understand the math but overall it was very straightforward and the math itself was very easy to implement as well.

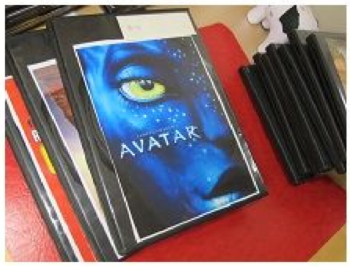
2.4 Results and Discussion

The results for my program, implementing the algorithm described by [2] had high success rate. Generally, if the DVD is upright with no occlusion, the success is very high, as a result, I removed those types of DVDs in my test set. For the DVDs that I did keep, I opted to choose the images that seemingly has occlusion, rotation, multiple DVDs in the picture and DVDs being held in a different orientation, they show great results as well. Some of the results are shown in the figures below…

```
query_dir =

dvd_covers/Palm/034.jpg


output_top10 =

   34.0000    1.8719
    9.0000    1.8879
   88.0000    1.8910
   92.0000    1.8913
   54.0000    1.8931
   48.0000    1.8932
   79.0000    1.8940
   25.0000    1.8962
   96.0000    1.8964
   69.0000    1.8977
```

```
query_dir =

dvd_covers/Canon/088.jpg


output_top10 =

   88.0000    1.8777
   54.0000    1.8877
   79.0000    1.8898
   22.0000    1.8914
   48.0000    1.8942
   28.0000    1.8964
    2.0000    1.8966
   77.0000    1.8980
   26.0000    1.8992
   25.0000    1.8995
```

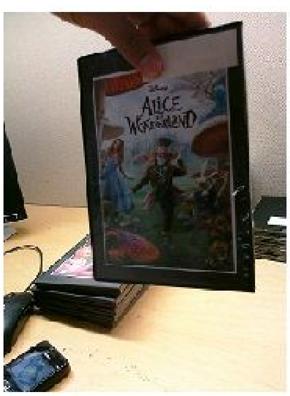
```
query_dir =

dvd_covers/E63/027.jpg


output_top10 =

   27.0000    1.8460
   25.0000    1.8866
   58.0000    1.8880
   43.0000    1.8899
    5.0000    1.8922
   54.0000    1.8926
    9.0000    1.8932
   96.0000    1.8936
   48.0000    1.8938
   24.0000    1.8954
```



```
query_dir =

dvd_covers/Palm/055.jpg


output_top10 =

   55.0000    1.8098
   53.0000    1.8815
   68.0000    1.8926
   24.0000    1.8942
   96.0000    1.8949
    8.0000    1.8969
   94.0000    1.8988
   92.0000    1.8988
   48.0000    1.8989
   35.0000    1.9002
```

```
query_dir =

dvd_covers/Droid/092.jpg


output_top10 =

   92.0000    1.8874
   54.0000    1.9041
   25.0000    1.9159
   32.0000    1.9173
   15.0000    1.9188
   93.0000    1.9210
   94.0000    1.9213
   30.0000    1.9237
   61.0000    1.9253
   10.0000    1.9255
```

```
query_dir =

dvd_covers/Droid/099.jpg


output_top10 =

   99.0000    1.8587
   25.0000    1.8957
   24.0000    1.8960
    2.0000    1.8961
   83.0000    1.8962
   96.0000    1.8967
   21.0000    1.8972
   54.0000    1.8982
   71.0000    1.8990
   27.0000    1.8998
```



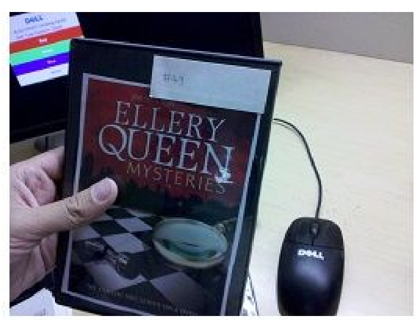


Figure 1 - Running just the hierarchical scoring algorithm

```
query_dir =

dvd_covers/Droid/092.jpg


output_top10 =

   68.0000    1.8493
   92.0000    1.8658
   21.0000    1.8784
   70.0000    1.8903
   53.0000    1.9014
   55.0000    1.9061
   35.0000    1.9071
   54.0000    1.9105
   33.0000    1.9111
    5.0000    1.9112


best_img =

    92
```

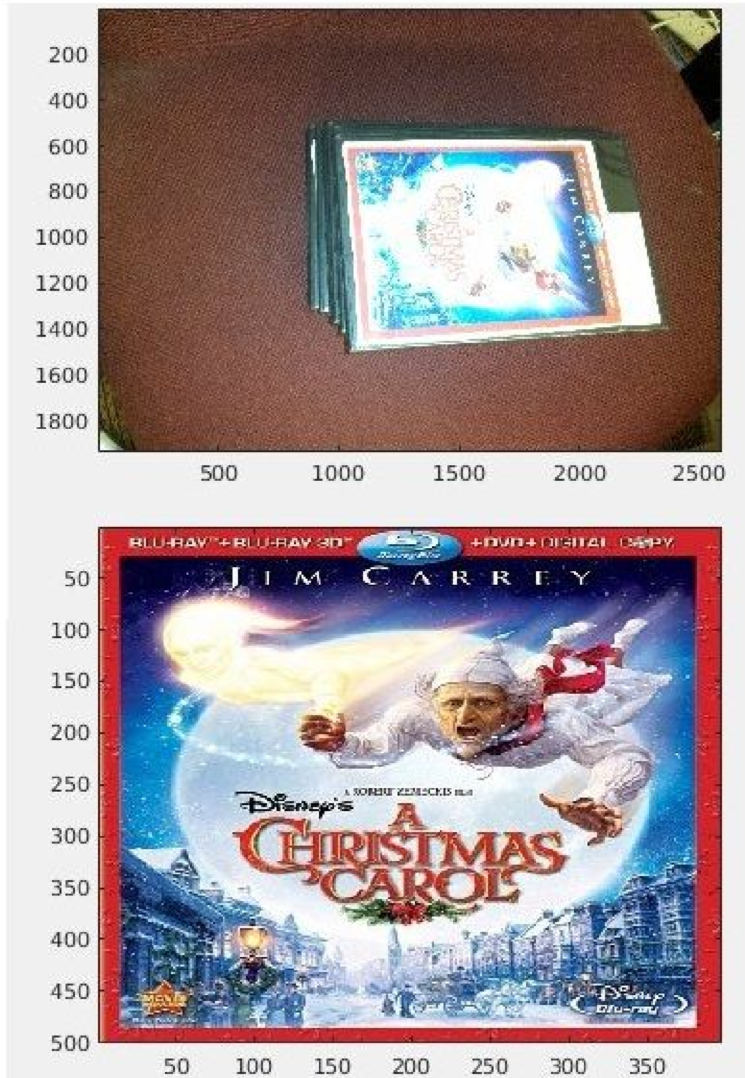


Figure 2: Ransac on top 10 images

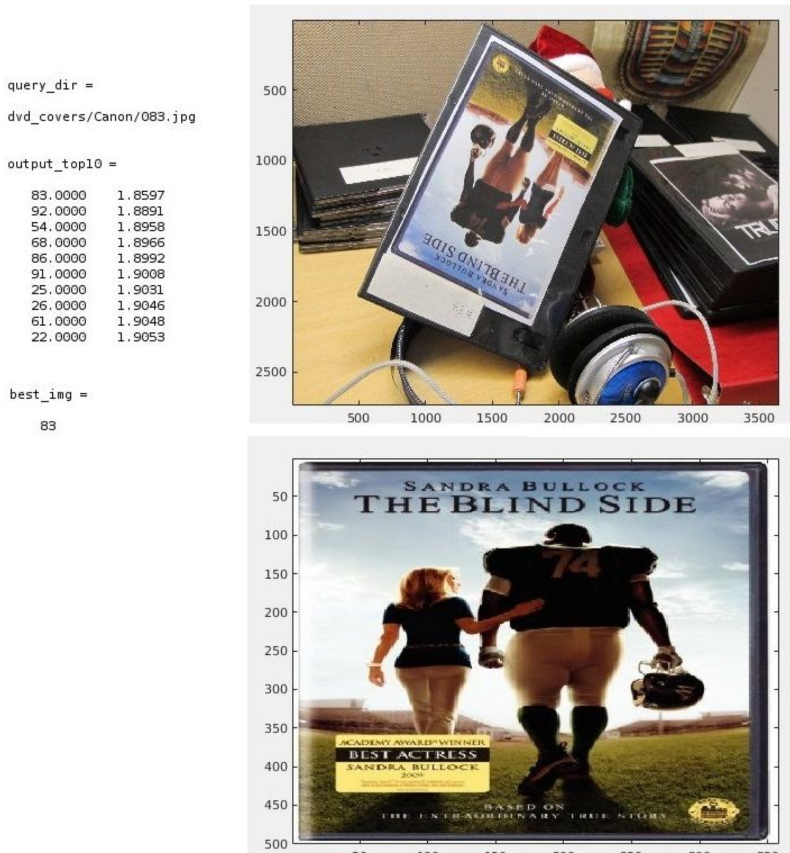


Figure 3: Ransac on top 10 images

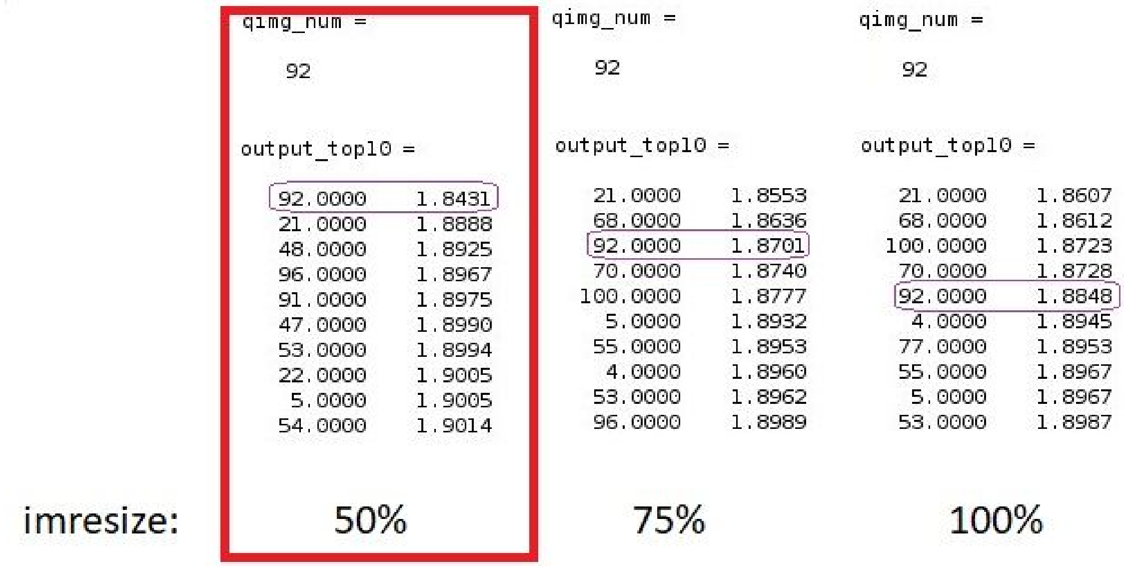
The DVDs that generally don't place number 1 in the results are DVDs that seem to have the cover distorted or partially covering glare on the dvd image. However, they appear in the top 10 scores outputted by our hierarchical scoring algorithm, and we can use Ransac to correctly find the image from the top 10 results.

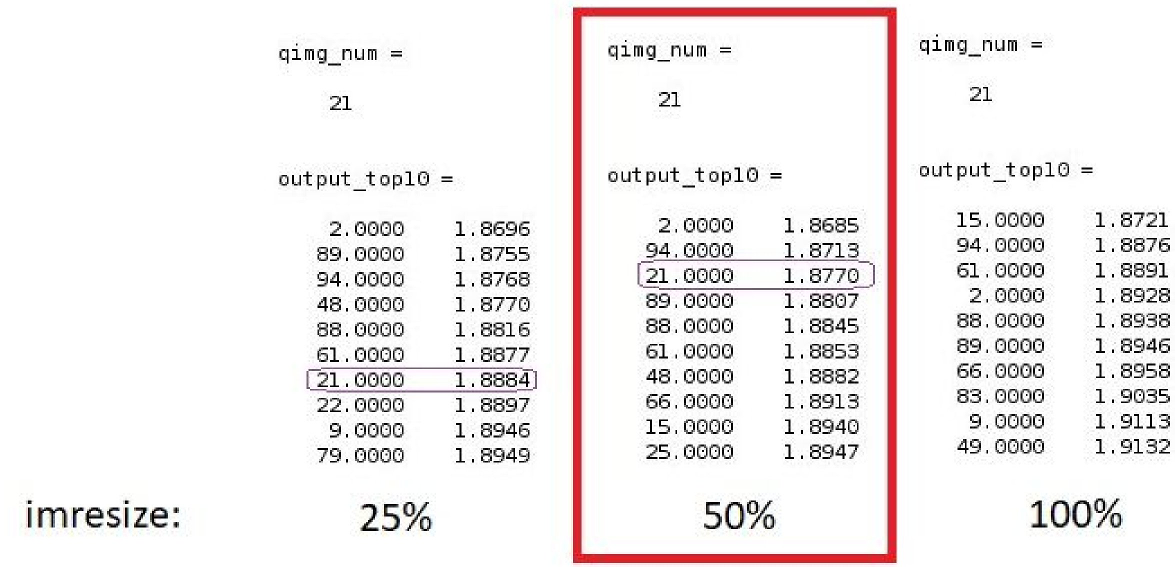
2.4.1 Space and runtime complexity

The algorithm needed to store $O(K^L)$ cluster centers for the tree. Since my implementation does precomputing of the image and query vectors, it also needed to store $O(N*(K^L))$ nodes for N+1 images (db images and query images). The runtime of the algorithm was not too bad. One of the main advantages of [2] was the runtime since a given image only needed to traverse from the root to a leaf instead of iterating through all the leaves. As a result, creating the vocabulary tree and training only took a few minutes to complete.

2.4.2 Improvements

Two improvements I have made to improve the results from my programming that implements [2] are cumulative weights and resizing the query image. How cumulative weights works in my implementation is that by accumulating the weight scores w_i of nodes in the top parts of the tree and propagating down the scores to the leaf nodes. This makes it so that the nodes near the bottom are weighted more heavily than the top. Thus database images that have a similar node path near the bottom nodes with a given query image, will score much higher in my implementation. Another improvement was resizing the query image. Since the query image are images that are taken from various cameras, they do not have a fix size. Some images also have occlusion and other background objects in the picture. Resizing the query image may scale down the amount of “background” in the query image. Thus it will decrease the amount of noise in the image, and has better features extracted when we use SIFT.





Resizing the picture by 50% typically has the best effect.

2.5 Conclusion and future work

Overall, the experience of implementing [2] has been great. In contrast to what we were taught in class about inverted file indexes and TF-IDF scoring, I have gained new insight on a new way of classifying images based on words using a vocabulary tree and computing the scores to find the best match. The VLFeat library has another implementation of feature extraction which can extract affine invariant features. In my future work to add onto this algorithm, it may be possible that using such features may improve the results of my implementation even more.

2.6 References

[1] http://www.vlfeat.org/overview/sift.html

[2] http://www-inst.eecs.berkeley.edu/~cs294-6/fa06/papers/nister_stewenius_cvpr2006.pdf

[3] http://www.vlfeat.org/matlab/vl_hikmeans.html

[4] https://sites.google.com/site/chenmodavid/datasets