Project 4 : Scalable Recognition with a Vocabulary Tree

Min Hyeok Lee – 1004940273 (leemin44)

1. Introduction

This project is based on the paper by Nister, Stewnius, Scalable Recognition with a Vocabulary Tree, CVPR 2006[1]. Given test image, it retrieves best matching DVD cover from the database and visualize its homography.

2. Method and code

2.1 Vocabulary Tree

```
class VTree:
    def __init__(self, k, L):
        self.des_counts = {} #records the number of descriptor vectors of image in each node
        self.num_img = 0 #number of imgs in the database
        self.all_img = [] #list of all imgs
        self.des_pair = [] #record of one descriptors and label with its image name
        self.tree = None
        self.num_leaf = 0 #also used to count index of node
        self.weights = None
        self.k = k
        self.L = L
        self.database_vectors = {}
```

- It is a database which stores the descriptors of all images in “DVDcovers”. The tree has branching factor K, which is also used in K-mean with level, L.

```
    #load the img from the database "./DVDcovers"
    def load_imgs(self, path):
        for filename in os.listdir(path):
            if "jpg" in filename:
                img_path = os.path.join(path, filename)
                img = cv2.imread(img_path)
                kp, des = extract_features(img)
                self.num_img += 1
                self.all_img.append(filename)
                self.des_counts[filename] = np.zeros(self.k**self.L)
                #self.database_vectors[filename] = np.zeros(self.k**self.L)
                for descriptor in des.tolist():
                    self.des_pair.append((descriptor, filename))
        self.weights = np.zeros(self.k**self.L) #number of leaf nodes

    def build_tree(self, k, L, pairs):
        all_descriptors = [tup[0] for tup in pairs]
        node = Node()
        node.kmeans = KMeans(n_clusters = k, random_state=0)
        node.kmeans.fit(all_descriptors)

        if L == 0: # leaf node now count the number of descriptors of img and save
            node.index = self.num_leaf #node index, i..
            for tup in pairs:
                img = tup[1]
                if img not in node.num_des:
                    node.num_des[img] = 1
                else:
                    node.num_des[img] += 1

            # increment the number of descriptor vector (m_i) of each image count dictionary
            for img, count in node.num_des.items():
                self.des_counts[img][node.index] += count

            #for each node save weights w_i = ln(N/N_i)
            self.weights[node.index] = np.log(self.num_img/len(node.num_des))
            self.num_leaf += 1
            return node
        else: #not leaf node so run the KMean again until it reaches leaf
            for i in range(k): #divide by branching factor k
                p = np.array(pairs, dtype=object)
                labels = np.array(node.kmeans.labels_)
                cluster = p[labels == i]
                child = self.build_tree(k, L-1, cluster)
                node.children.append(child)
        return node
```

After extracting features from the images in database, all descriptors for each DVD cover is used to build the tree by running K-means with 'k' value given. The group of descriptors will be partitioned into 'k' group, then for each group, we run the K-mean and partition recursively until we reach the 'L' levels. The clusters in the leaf node act as word.

2.2 Query image, database image vector

$$\begin{aligned} q_i &= n_i w_i \\ d_i &= m_i w_i \end{aligned} \qquad w_i = \ln \frac{N}{N_i},$$

Query image vector (q_i) is the product of weight of node i and number of descriptors of query image. It is computed by "comp[ute_query_vector()" function in the code.

```
def compute_query_vectors(self, query_img):
    img = cv2.imread(query_img)
    kp, des = extract_features(img)

    n = np.zeros(self.num_leaf, dtype=int)
    target_imgs = []
    for d in des:
        node = self.get_leaf_node(self.tree, d)
        for img, count in node.num_des.items():
            if img not in target_imgs:
                target_imgs.append(img)
        n[node.index] += 1
    #compute q_i = n_i * w_i
    q = np.zeros(self.num_leaf)
    for w in range(len(self.weights)):
        q[w] = n[w]  * self.weights[w] #n_i * w_i
    q = q / np.sum(q)
    return q, target_imgs
```

After extracting features of query image, and for each descriptor vector, we traverse down the vocabulary tree by calculating the euclidean distance between descriptor vector and tree node's cluster centers (centroid). Once it reaches the leafe node, it records all images in the leaf node and also calculates the 'q_i' given weights and number of descriptor vectors that belongs to the query image.

The weight (w_i) is calculated in the "build_tree()" by dividing number of all database images and number of images in the node then take a log of it.

```
def compute_distance_vectors(self):
    for img in self.all_img:
        d = (self.des_counts[img] / np.sum(self.des_counts[img])) * self.weights
        self.database_vectors[img] = d
```

Database image vector (m_i) is the product of weight of node i and number of descriptor of database image.

Both vectors are normalized by dividing each vector by sum of all vectors.

2.3 Compute score

$$
\begin{aligned}
\| \mathbf{q} - \mathbf{d} \|_p^p &= \sum_i |q_i - d_i|^p & (5) \\
&= \sum_{i|d_i=0} |q_i|^p + \sum_{i|q_i=0} |d_i|^p + \sum_{i|q_i\neq 0, d_i \neq 0} |q_i - d_i|^p \\
&= \|\mathbf{q}\|_p^p + \|\mathbf{d}\|_p^p + \sum_{i|q_i\neq 0, d_i \neq 0} (|q_i - d_i|^p - |q_i|^p - |d_i|^p) \\
&= \quad 2 + \sum_{i|q_i\neq 0, d_i \neq 0} (|q_i - d_i|^p - |q_i|^p - |d_i|^p),
\end{aligned}
$$

```
def compute_score(self, query_vectors, target_imgs):
    score = np.zeros(len(target_imgs))
    for idx in range(len(target_imgs)):
        img = target_imgs[idx]
        d_i = self.database_vectors[img]
        #since we used L1 normalization for each vector we use eq.5 from the paper.
        score[idx] = 2 + np.sum(np.abs(query_vectors - d_i) - np.abs(query_vectors) - np.abs(d_i))
    return score
```

Using the scoring method defined in the paper, we take in the two descriptor (q_i, d_i) for query and database image. In the paper, David stated that L-1 normalization gives better results than L-2 so we used above equation, (5) to calculate the score between query images and database images. Then by using sorting method, retrieved 10 lowest score for the RANSAC method.

2.4 RANSAC, Homography, SIFT feature extractor

I have reused the code that I have implemented in assignment 4 of this course for RANSAC and homography to get the number of inliers of 10 images with lowest score. All codes related to this section is in "helper.py". To get the number of inliers, I calculated the Euclidean distance between two matched points by transforming one point then if distance is less than 35, I incremented number of inliers. Since there are not many DVD cover image in database, all top 10 image may have may words in common. Then by running RANSAC I retrieved the best image with highest number of inliers.

3. Test (how to run script):

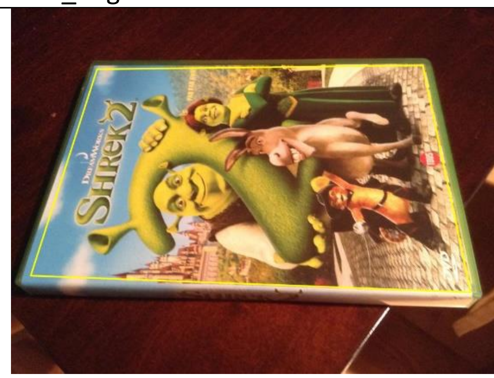
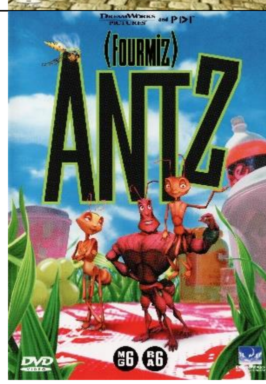
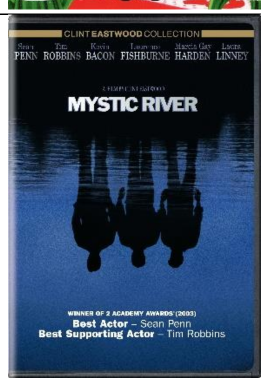
```
#To test one img
#test_imgs = ["./test/image_01.jpeg"]
#Initialize database
vt = VTree(5,5)

#uncomment this if you want to create new database with different k and L
'''
#load imgs and build vocabulary tree using k, and L given
vt.load_imgs(dir_path)
vt.tree = vt.build_tree(vt.k, vt.L, vt.des_pair)
#save database
vt.save("database{}{}.txt".format(vt.k, vt.L)
'''

#if database already, load database
print("loading database")
'''
load database with k = 5, L = 5
'''
vt.load("database55.txt")
'''
load database with k = 6, L = 4
'''
#vt.load("database64.txt")
```

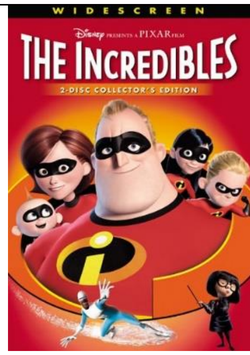
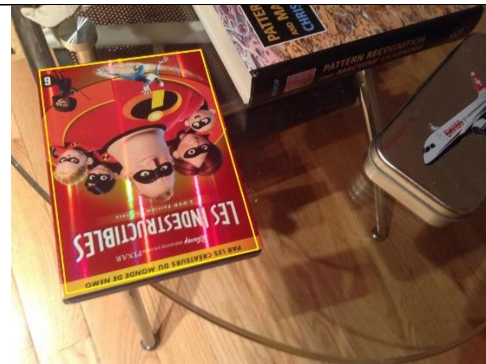
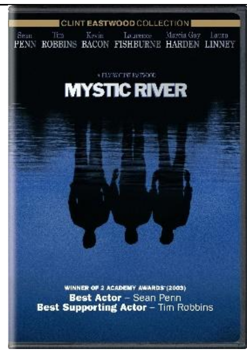
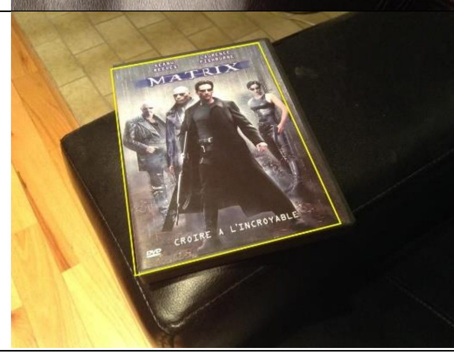
Initialize VTree with k and L and it will initialize the tree then save it into "databasekl.txt".The file "database55.txt" is the database with k = 5 and L = 5, and "database64.txt" is the database with k = 6 and L = 4. Then running the "Vocabulary Tree" will automatically compute the process of retrieving and find the best image.

Result 1 shows the output of using "database64.txt" which uses k = 6 and L = 4 and Result 2 shows the output of using "database55.txt" which uses k = 5 and L =5. In Result 1, all DVD covers were matched except fort test2 and test 6. Test 6 image is not found in the database so the DVD cover with highest number of inliers were returned which is "Mystic River". Also test 2 image was not able to identify its best match image correctly. However, in result 2, test2 image was correctly detected whereas test 5 image failed to do so.

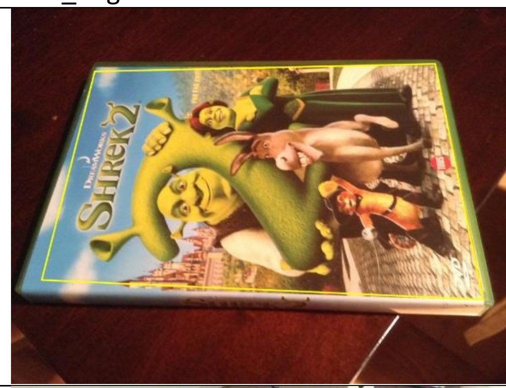
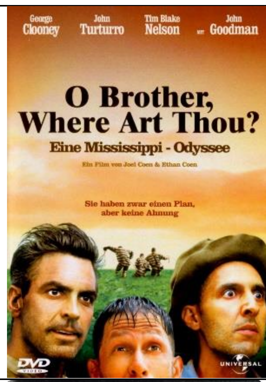
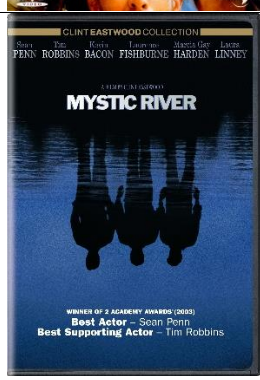
4. Results

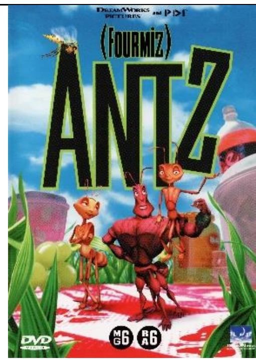
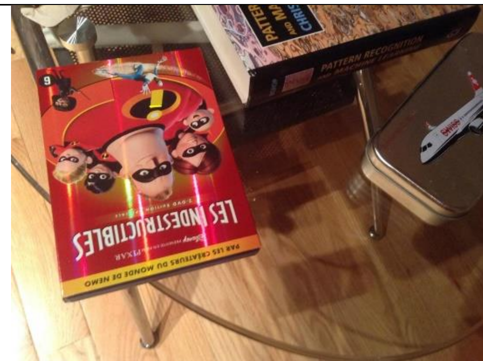
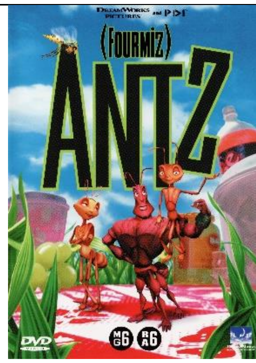
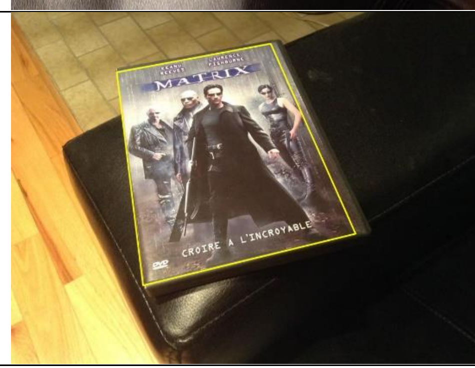
Result1 – running the test images with database k = 6/ L =4

Database K6,L4	DVDcover	Test_img
1		
2		
3		
4		

5	WIDESCREEN THE INCREDIBLES 2-DISC COLLECTOR'S EDITION	LES INDESTRUCTIBLES
6	CLINT EASTWOOD COLLECTION Sean PENN Tim ROBBINS Kevin BACON Laurence FISHBURNE Marcia Gay HARDEN Laura LINNEY MYSTIC RIVER WINNER OF 2 ACADEMY AWARDS (2003) Best Actor – Sean Penn Best Supporting Actor – Tim Robbins	Toronto
7	KEANU REEVES LAURENCE FISHBURNE MATRIX	MATRIX CROIRE A L'INCROYABLE

Result2 – running the test images with database k = 5/ L =5

Database K5,L5	DVDcover	Test_img
1		
2		
3		
4		

5		
6		
7		

5. Conclusion

Some images were correctly retrieved from the database. The reason behind this may be either the matching DVD cover does not exist in the database or Since the database is not large enough to distinguish the DVD covers. Retrieving top 10 have lots of common visual words therefore there are not much difference between scores of targeted images. Therefore, some DVD cover retrieval may be wrong. If we retrieve more than 10 in the targeted image such as 20~30 the program correctly retrieves the best matching DVD cover.

6. Reference

[1] David et al., Scalable Recognition with a Vocabulary Tree.