Table of Contents

Steps followed in Algorithm 2

All Results for one of the case 3

1.) PCA\_SIFT Features 3

2.) K-means. 4

3.) Create histograms using codebook. 4

4.) KNN and predicting 5

5.) Final step – Accuracy 5

6.) Proof of correctness 6

Results with various combinations 6

Results in summary (tabular form) 12

Comments 12

Final code (with comments) 13

# Steps followed in Algorithm

# All Results for one of the case

Let’s take dummy case in which I have deliberately trimmed down the parameters so as for easy visualization.

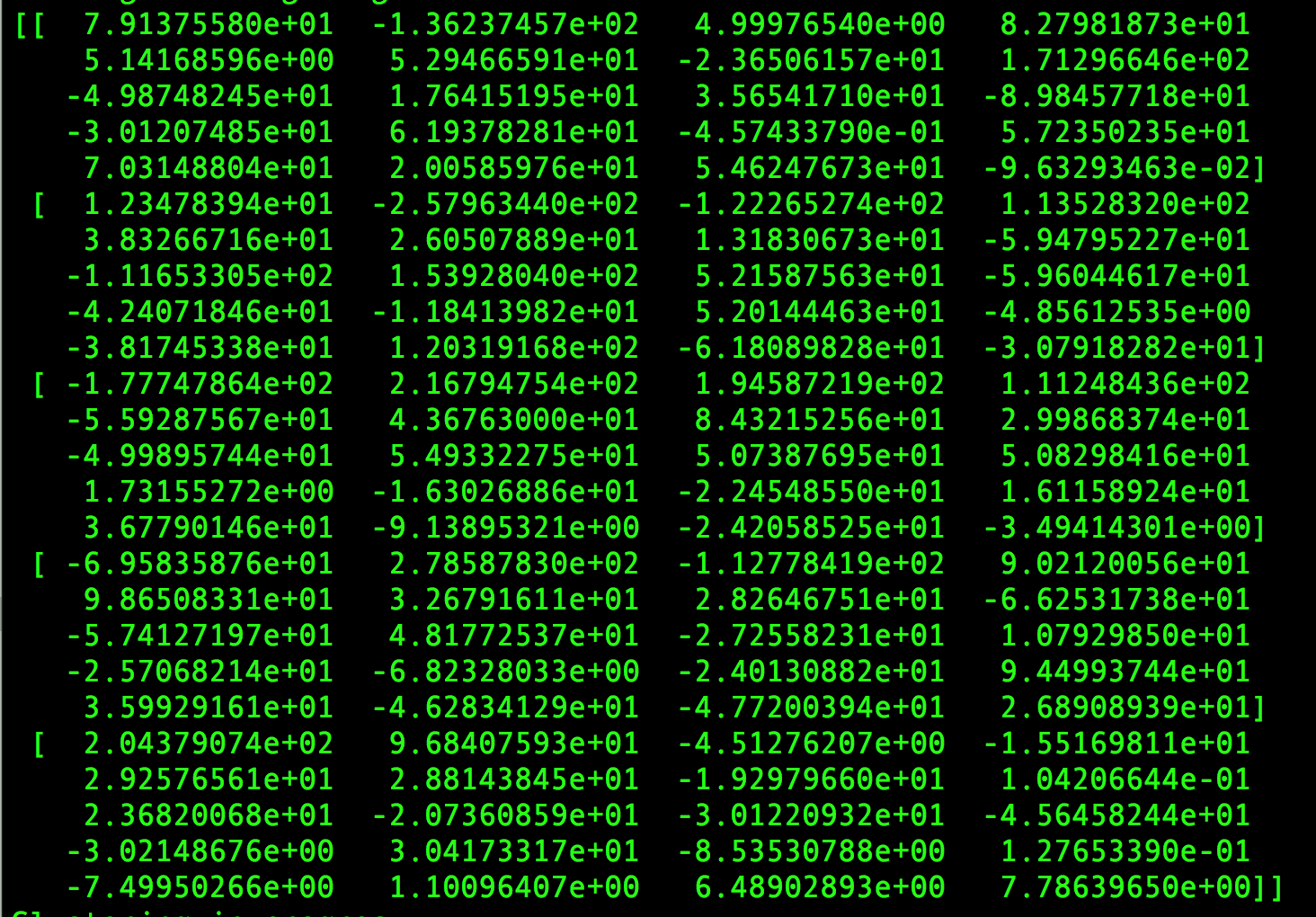
Consider the following configurations:

PCA Dimension: 20

Number of k clusters: 8

Number of kNN neighbours: 10

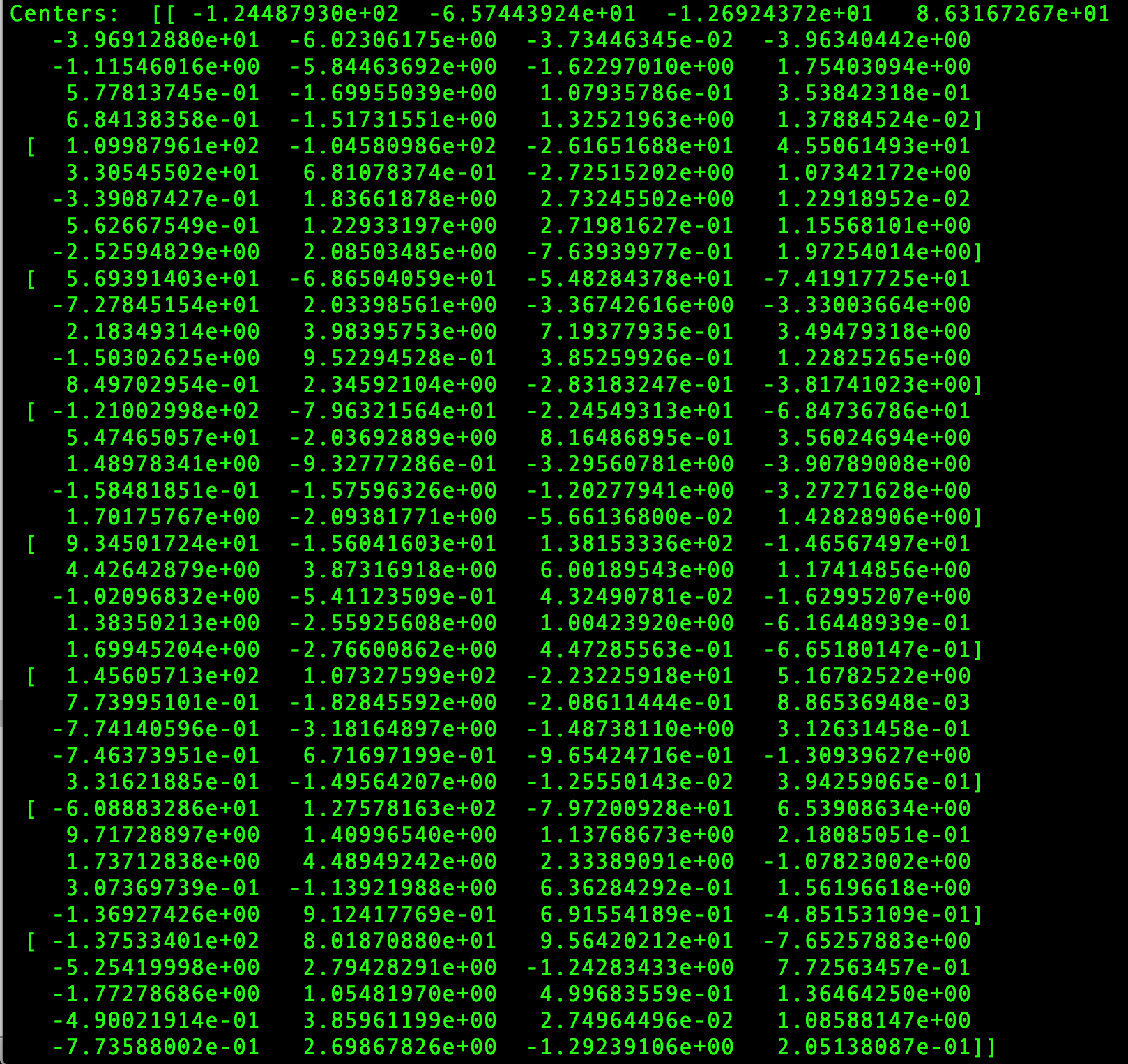
## 1.) PCA\_SIFT Features

First of all, the entire training set of images are read. SIFT is applied on all these images one by one. PCA is applied on top of these SIFT descriptors , while stacking PCA progressively. The idea is to combine data set for k-means algorithm.

The above figure shows pca\_shift\_features of airplanes/train/image\_0071.jpeg

## 2.) K-means.

Next, we apply on the stack of pca\_sift\_features formed in the step above. Below are the centers we find. We find 8 centers each of dimension 1X 20.



## 3.) Create histograms using codebook.

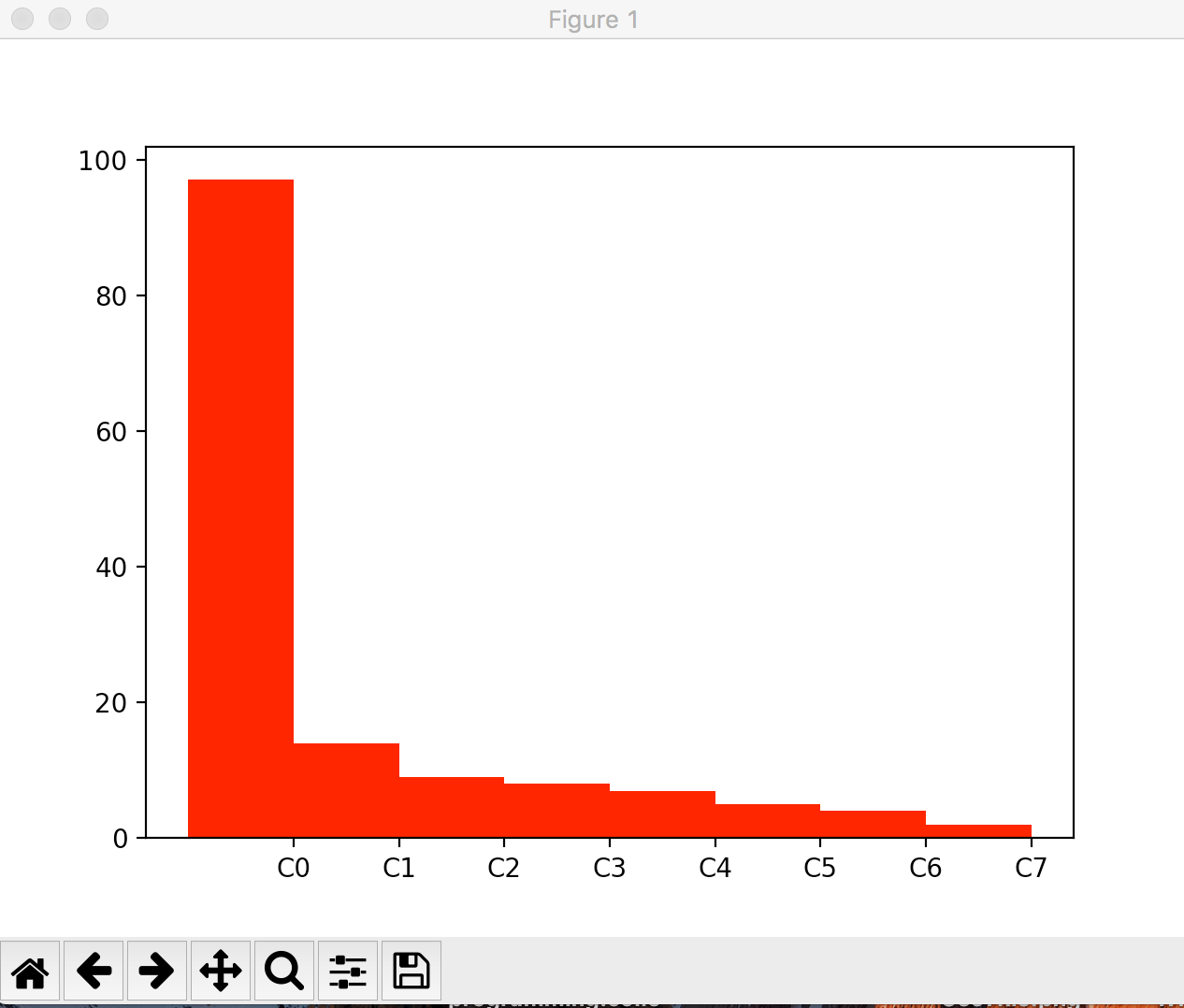
In this step, basically we create histogram using image features and codebook.

Codebook is the assigned codes to k clusters by algorithm. Intention is to find

what is frequency of cluster supported by respective descriptor of an image.

Here is the sample of histogram example of one of the training image. That is: airplanes/train/image\_0071.jpeg

Histogram achieved for this image was [97, 14, 9, 8, 7, 5, 4, 2]. Such that value at each ith index is the frequency of cluster ‘i’ supported by feature of the above mentioned training image. It can be visualized as below image



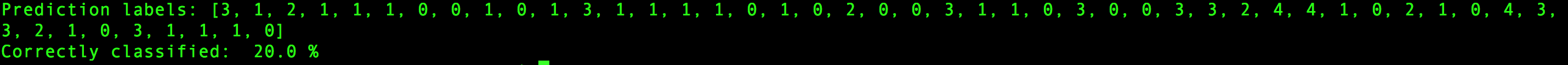
## 4.) KNN and predicting

Last step is to do kNN analysis on histograms achieved in the step above and next is to predict labels.

Prediction achieved for 50 test images is given as:

[0, 0, 1, 4, 1, 0, 1, 3, 3, 3, 1, 2, 2, 1, 2, 0, 1, 2, 3, 0, 0, 1, 3, 3, 0, 0, 4, 1, 1, 2, 2, 3, 1, 3, 1, 0, 1, 0, 0, 3, 0, 2, 0, 0, 3, 3, 1, 3, 1, 4]

Screenshot:



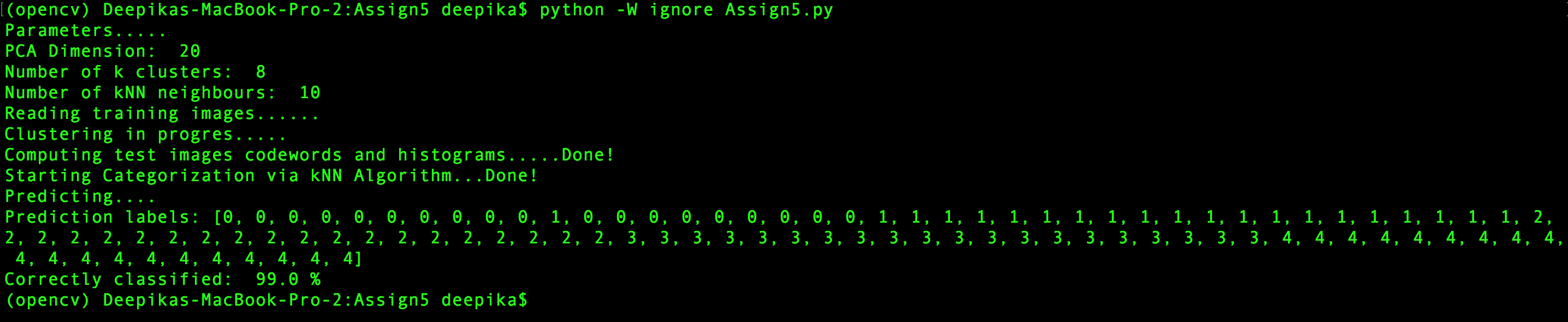
## 5.) Final step – Accuracy

The last step of the algorithm is to compute accuracy. In this case, since clusters were very small hence accuracy is 20.0% (Screenshot above. )

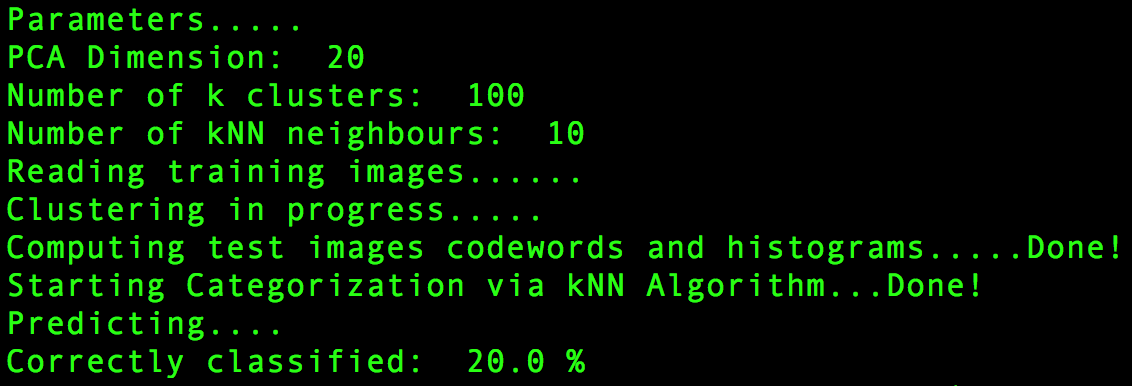
## 6.) Proof of correctness

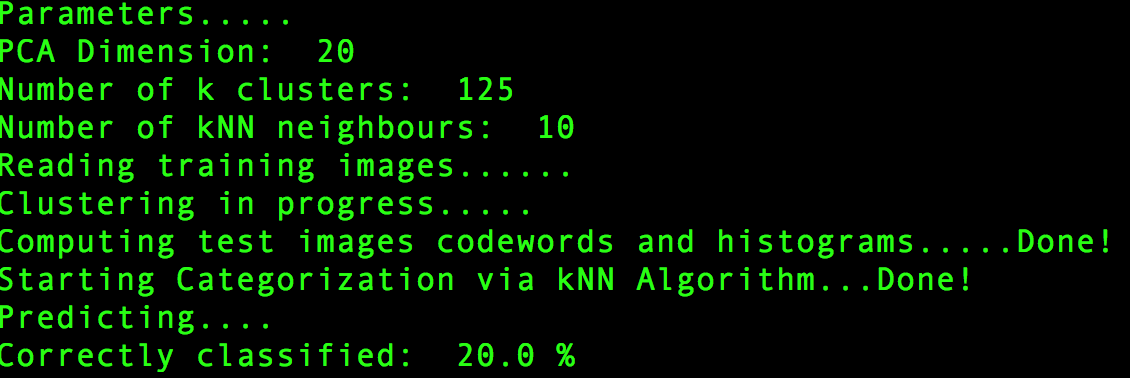
When I ran the algorithm on various combinations of key parameters that is PCA dimension, K clusters, number of neighbors in kNN, I found that accuracy was 100% in most cases. Even with such a few number of clusters, achieved accuracy by using training dataset as test dataset is 99%.

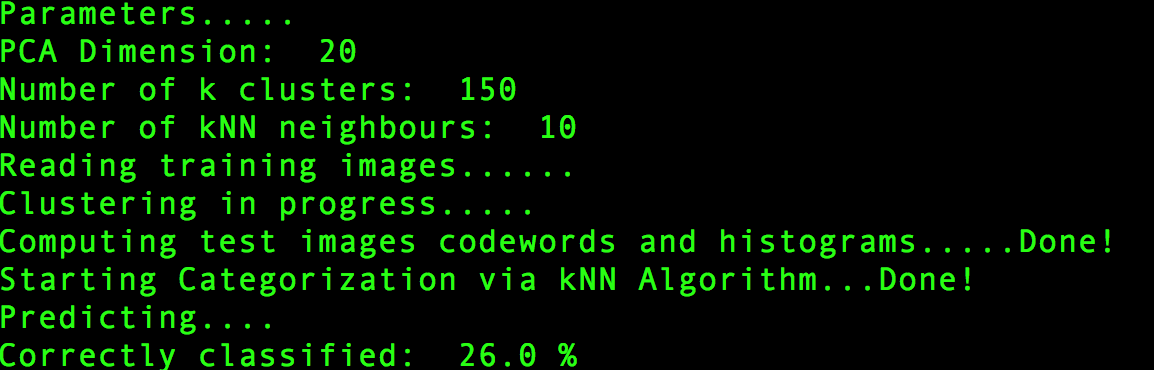
Screenshot:

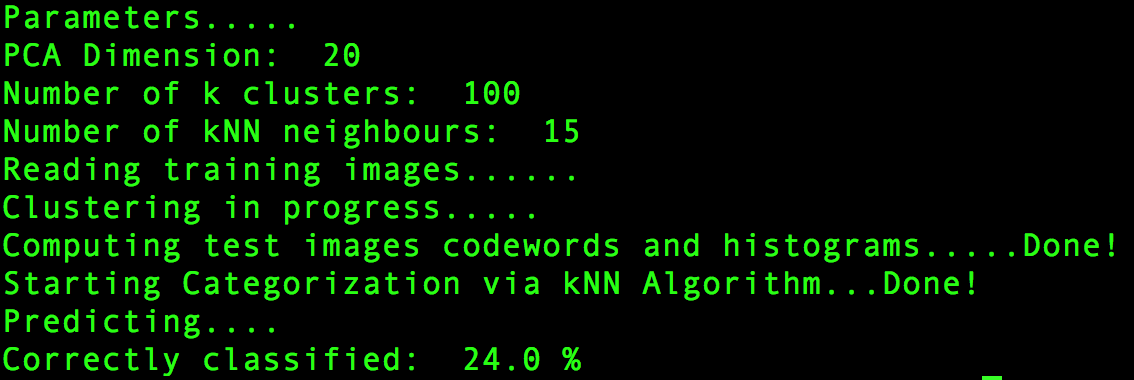


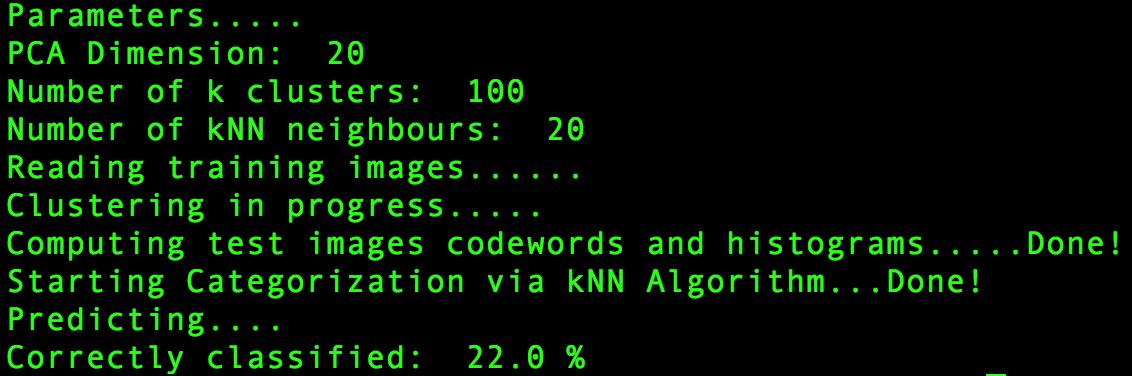
# Results with various combinations

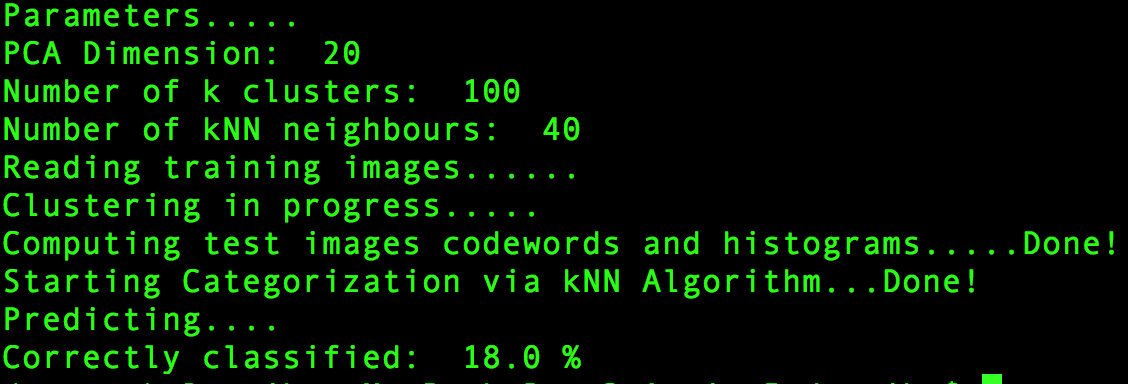


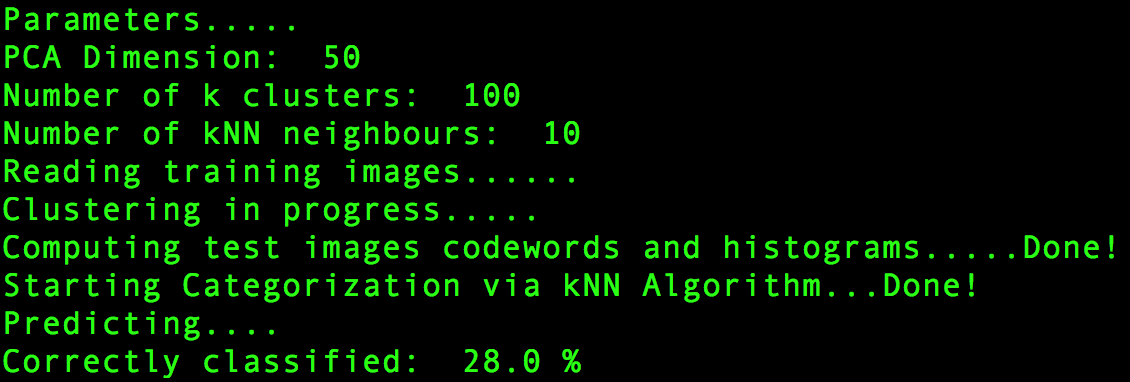


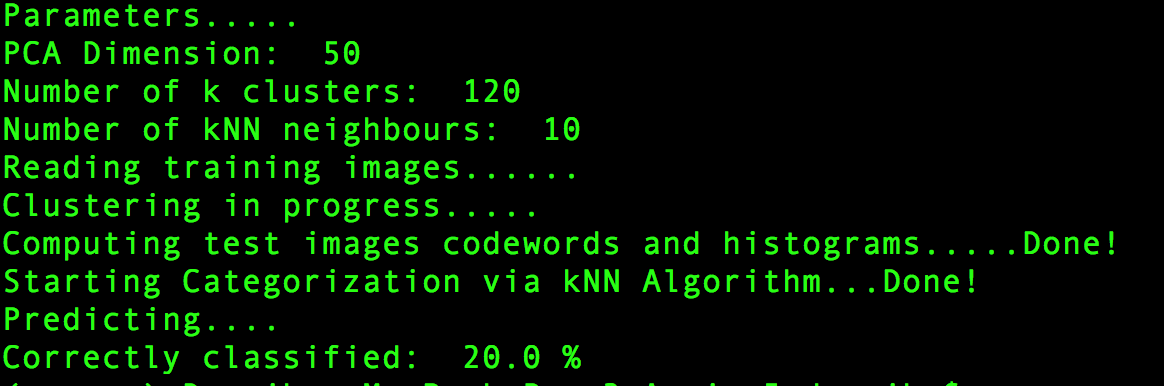


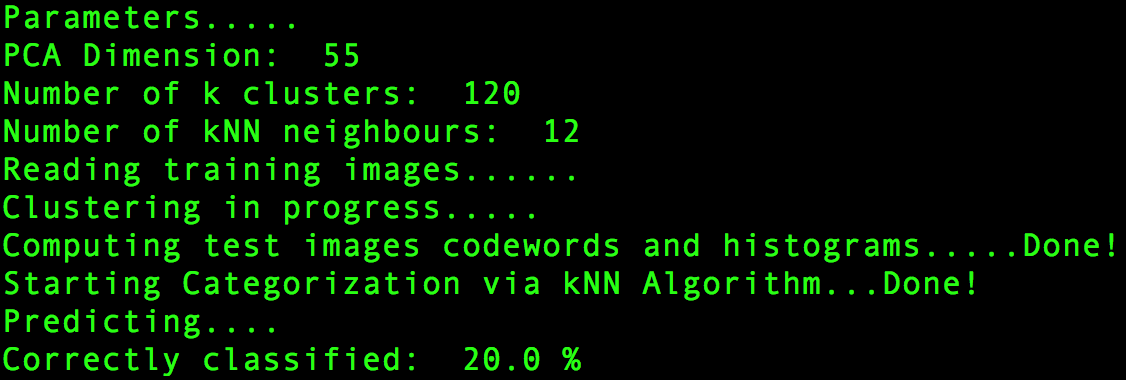


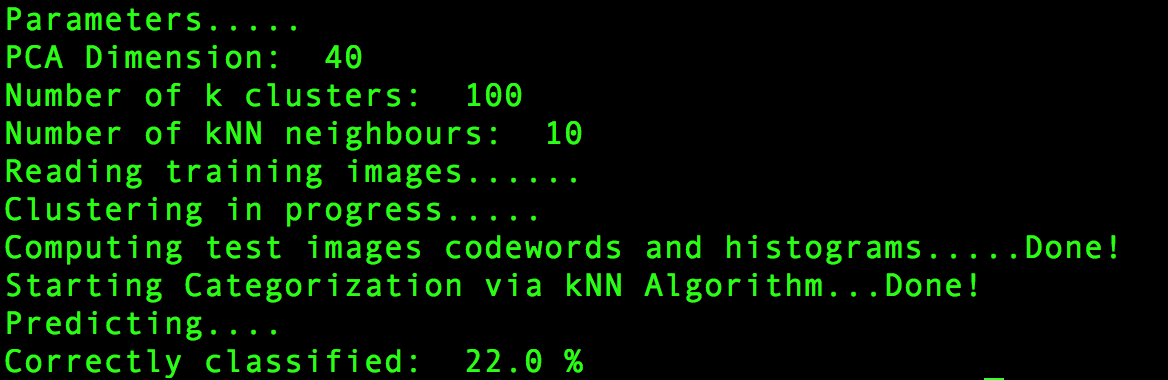


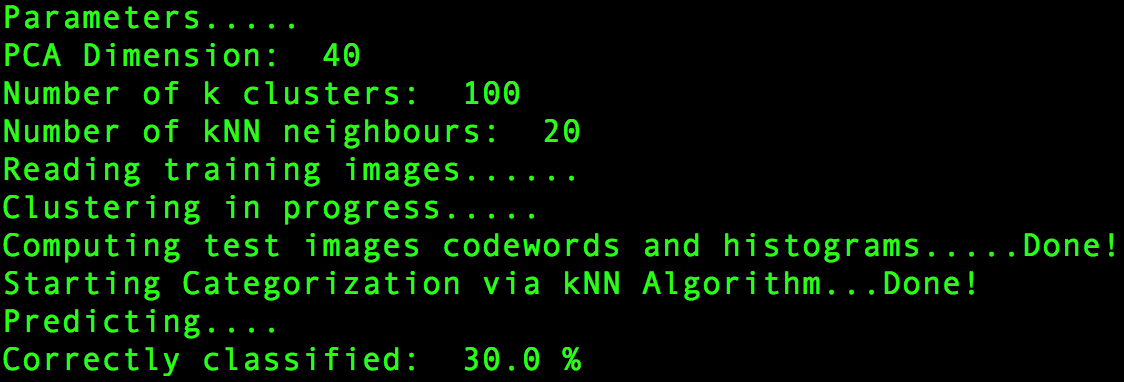


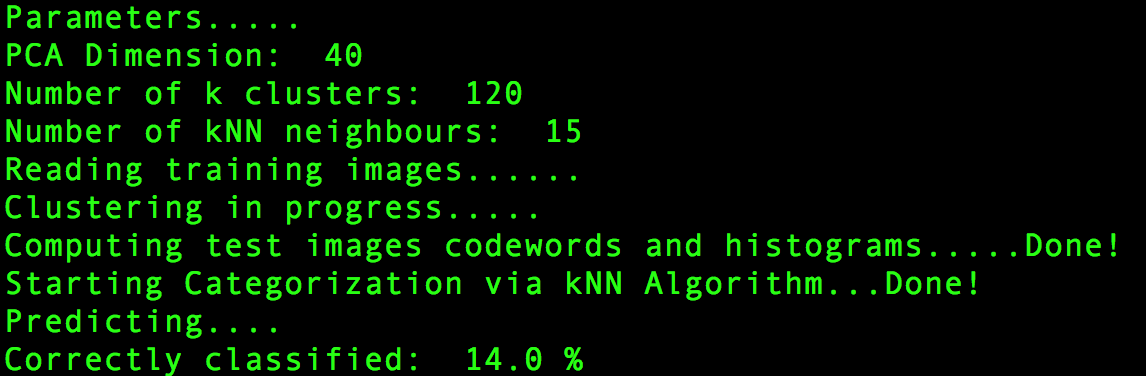


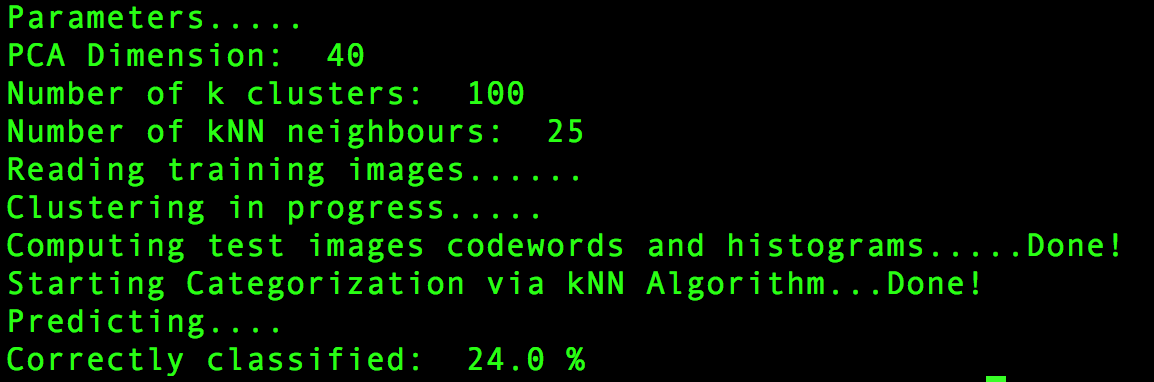


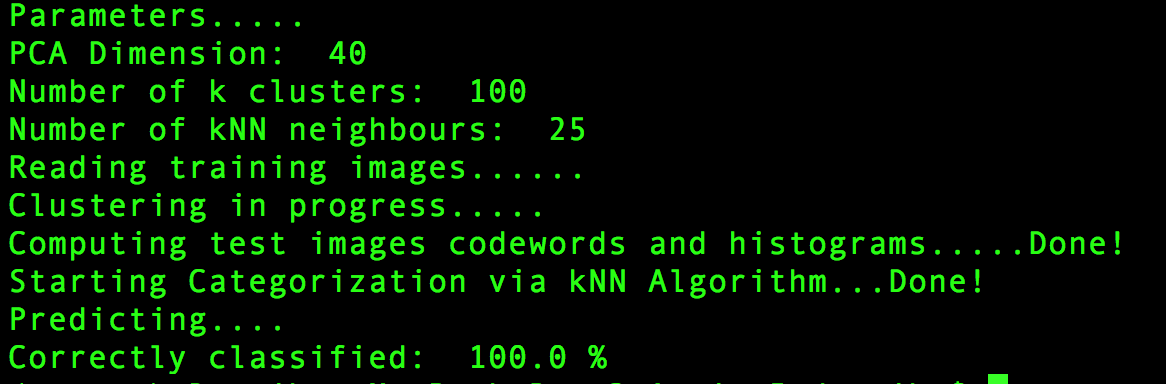












# Results in summary (tabular form)

|  |  |  |  |
| --- | --- | --- | --- |
| PCA Components | K clusters | Neighbors in kNN | Accuracy |
| 20 | 100 | 10 | 20% |
| 20 | 125 | 10 | 20% |
| 20 | 150 | 10 | 26% |
| 20 | 100 | 15 | 24% |
| 20 | 100 | 20 | 22% |
| 20 | 100 | 40 | 18% |
| 50 | 100 | 10 | 28% |
| 50 | 120 | 10 | 20% |
| 55 | 120 | 12 | 20% |
| 40 | 100 | 10 | 22% |
| 40 | 100 | 20 | 30% |
| 40 | 120 | 15 | 14% |
| 40 | 100 | 25 | 24% |
| 40 | 100 | 25 | 100%  (Using training data for testing) |

# Comments

The algorithm accuracy varies depending on centers chosen by k-means. the goodness of center decides the accuracy of the algorithm . The best accuracy I could achieve was using PCA components = 40, clusters = 100 and k-nearest neighbors as 20. Each of these parameters have peek. That is increasing each of the key parameters till a point increased accuracy and then suddenly drops down. So, a lot of tuning and trial/error analysis is required to come up with high accuracy. This is one of disadvantage of this algorithm as parameters need to be tuned depending on input data set.

K-means cannot handle non-linear functions. So, if any of the images had a non-linear function that would have been less favored in predictions.

# Final code (with comments)

\_\_author\_\_ = **'deepika'  
  
import** cv2  
**import** matplotlib  
**import** sys  
  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**from** os **import** listdir  
  
**"""  
Global Variables  
"""**\_\_CATEGORIES\_\_ = 5  
categories = [**"airplanes"**, **"car\_side"**, **"electric\_guitar"**, **"faces"**, **"Motorbikes"**]  
PCA\_COMPONENTS = 40  
NUMBER\_OF\_CLUSTERS = 100  
NUMBER\_OF\_NEIGHBOURS = 25 *#not less than 10***"""  
 Function used to convert image to gray scale  
"""  
def** to\_gray(color\_img):  
 gray = cv2.cvtColor(color\_img, cv2.COLOR\_BGR2GRAY)  
 **return** gray  
  
**"""  
 Function used to generate SIFT features of an image  
"""  
def** gen\_sift\_features(gray\_img):  
 sift = cv2.xfeatures2d.SIFT\_create()  
 kp, desc = sift.detectAndCompute(gray\_img, None)  
 **return** kp, desc  
  
**def** do\_k\_means(Z):  
 *# Define criteria = ( type, max\_iter = 10 , epsilon = 1.0 )* criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)  
  
 *# Set flags (Just to avoid line break in the code)* flags = cv2.KMEANS\_PP\_CENTERS  
 *#KMEANS\_PP\_CENTERS; KMEANS\_RANDOM\_CENTERS* compactness, labels, centers = cv2.kmeans(Z, NUMBER\_OF\_CLUSTERS, None, criteria, 10, flags)  
 **return** labels, centers  
  
*#Given one feature***def** \_get\_codeword\_histogram(features, centers):  
 histogram = [0.0 **for** i **in** range(len(centers))]  
 **for** i **in** range(len(features)):  
 minDist = sys.maxint  
 cluster\_assigned = None  
  
 **for** code **in** range(len(centers)):  
 dist = np.linalg.norm(features[i] - centers[code])  
 **if** dist < minDist:  
 minDist = dist  
 cluster\_assigned = code  
  
 histogram[cluster\_assigned] += 1  
  
 **for** i **in** range(len(histogram)):  
 histogram[i] = histogram[i] \* 1.0 /len(features)  
 **return** np.array([histogram], dtype=np.float32)  
  
**"""  
 Given the folder token i.e testOrTrain,  
 this function reads all images in that folder recursively,  
 apply SIFT on individual images, PCA and  
 stacks PCA results vertically(later used) k-means clustering  
"""  
def** \_get\_required\_image\_features(testOrTrain=**"train"**):  
 image\_features = []  
 pca\_sift\_combined = []  
  
 **for** cat **in** categories:  
 mypath = **'./HW5\_Data/'** + cat + **"/"** + testOrTrain + **"/"** files = [f **for** f **in** listdir(mypath)]  
 **for** \_f **in** files:  
 **if** \_f == **'.DS\_Store'**:  
 **continue** image\_path = mypath + \_f  
 im = cv2.imread(image\_path)  
 source\_gray = to\_gray(im)  
 kp, desc = gen\_sift\_features(source\_gray)  
 meanResult, eigenvectors = cv2.PCACompute(desc, np.mean(desc, axis=0).reshape(1,-1), maxComponents = PCA\_COMPONENTS)  
 X2 = cv2.PCAProject(desc, meanResult, eigenvectors)  
 **if** len(pca\_sift\_combined) == 0:  
 pca\_sift\_combined = X2  
 **else**:  
 pca\_sift\_combined = np.vstack((pca\_sift\_combined, X2))  
 image\_features.append(X2)  
 **return** pca\_sift\_combined, image\_features  
  
**"""  
 This function actually handles prediction using wighted average  
 given neighbours and distances as inputs.  
"""  
def** \_weighted\_average(neighbours, dist):  
 prediction = []  
 **for** i **in** range(len(neighbours)):  
 votes = [0 **for** \_ **in** range(len(categories))]  
 **for** j **in** range(len(neighbours[0])):  
 votes[int(neighbours[i][j])]+=1.0/dist[i][j];  
  
 label = max(votes)  
 predicted\_label = votes.index(label)  
 prediction.append(predicted\_label)  
  
 **return** prediction  
  
**def** draw\_histogram(trainHis):  
 **print "Drawing for "**, trainHis  
 alphab = [**'C'** + str(x) **for** x **in** range(len(trainHis))]  
 frequencies = [int(x) **for** x **in** trainHis]  
  
 pos = np.arange(len(alphab))  
 width = 1.0  
  
 ax = plt.axes()  
 ax.set\_xticks(pos + (width/2))  
 ax.set\_xticklabels(alphab)  
  
 plt.bar(pos, frequencies, width, color = **'r'**)  
  
 plt.show()  
 raw\_input(**">Hit Enter To Close...."**)  
 plt.close()  
  
**"""  
 Main function to start processing  
"""  
def** startPredication():  
 **print "Parameters....."  
 print "PCA Dimension: "**, PCA\_COMPONENTS  
 **print "Number of k clusters: "**, NUMBER\_OF\_CLUSTERS  
 **print "Number of kNN neighbours: "**, NUMBER\_OF\_NEIGHBOURS  
 knn = cv2.ml.KNearest\_create()  
 trainCategories = []  
 trainHis = []  
 testHis = []  
  
 *#Step 1:  
 # Read the trainig data set:  
 #  
 # Now this step will convert each image of training data set to gray scale and then  
 # find SIFT features -> Apply PCA -> return image wise PCA and PCA of all images  
 # stacked on top of each other such that there shape is (32742, 20).* **print "Reading training images......"** pca\_sift\_combined, image\_train\_features = \_get\_required\_image\_features(**"train"**)  
  
 *#Step 2:  
 # Do k-means.  
 # This step handles k-means on training data set.  
 # As per assignment, I have used PCA\_SIFT features of all images of training data set to  
 # implement k-means* **print "Clustering in progress....."** labels, centers = do\_k\_means(pca\_sift\_combined)  
  
 **assert** len(image\_train\_features) == 100  
  
 *#Step 3:  
 # Create Histogram:  
 # In this step, basically we create histogram using image features and codebook.  
 # Codebook are the assigned codes to k clusters by algorithm. Intention is to find  
 # what is frequency of cluster supported by respective descriptor of an image.  
 # In the first for loop, I find histogram for Training data set, in the second loop we  
 # find histogram of Testing data set.* **for** m **in** range(0, len(image\_train\_features)):  
 **if** len(trainHis) == 0:  
 trainHis = \_get\_codeword\_histogram(image\_train\_features[m], centers)  
 **else**:  
 trainHis = np.vstack((trainHis, \_get\_codeword\_histogram(image\_train\_features[m], centers)))  
  
 **if** len(trainCategories) == 0:  
 trainCategories = np.full((1, 1), m/20)  
 **else**:  
 trainCategories = np.vstack((trainCategories, np.full((1, 1), m/20)))  
  
 **print "Computing test images codewords and histograms.....Done!"** ignore, test\_image\_features = \_get\_required\_image\_features(**"train"**)  
 **for** m **in** range(0, len(test\_image\_features)):  
 **if** len(testHis) == 0:  
 testHis = \_get\_codeword\_histogram(test\_image\_features[m], centers)  
 **else**:  
 testHis = np.vstack((testHis, \_get\_codeword\_histogram(test\_image\_features[m], centers))) *#[50, #clusters]  
  
 #Step 4:  
 # This is the training step for KNN Algorithm.  
 # Once trainer is ready. We pass testHistogram to find N nearest neighbours.* knn.train(trainHis, cv2.ml.ROW\_SAMPLE, trainCategories)  
 **print "Starting Categorization via kNN Algorithm...Done!"** ret,result,neighbours,dist = knn.findNearest(testHis,k=NUMBER\_OF\_NEIGHBOURS)  
  
 *#Step 5:  
 # Prediction:  
 # In this step we Actually predict the label using weighted average function.  
 # I have used, neighbours and dist results of findNearest function for prediction.* **print "Predicting...."** count = 0  
 prediction = \_weighted\_average(neighbours, dist)  
 *#print "Prediction labels:", prediction* **for** i **in** range(len(prediction)):  
 **if** prediction[i]==i/(len(prediction)/\_\_CATEGORIES\_\_) :  
 count = count + 1  
  
 *#Step 6:  
 # Computing Accuracy:  
 # This step is the crux of all steps implemented above, as in this  
 # we get to know overall accuracy of algorithm* **print "Correctly classified: "**, 100\*float(count)/len(prediction), **"%"**startPredication()