CSCI 574 Hw#5 : Object Recognition using Bag of Features

Name: Manan Vyas
USC-ID : 7483-8632-00
Email: mvyas@usc.edu

Task: To train a Bag of Features recognizer using the set of training images provided and then recognize the trained objects from unknown images using a vote based K-Nearest Neighbour classifier.

Code implemented in: C++, OpenCV (Build in CodeBlocks|MinGW)

Discussing in details the results obtained for the following combination of input parameters;

```
int const kComponents = 30 ;

int const kClusters = 100 ;

int const kNearestNeighbors = 10;
```

- Training: To generate the codewords during the training phase, the following steps were implemented;

  1. SIFT Features : Local SIFT Features were determined using the SIFT::operator()[1] function in OpenCV. Once these 128 dimension features are obtained, their complexity was to be reduced by doing a Principal Component Analysis on these features. For this purpose, a fixed number of initial eigen vectors that correspond to the principal directions of these features were taken into account. I experimented with a varying number of these initial number of prinicipal direction (here taken to be 30) in addition to the recommended 20 components given in the problem. The dot product of the SIFT features and the principal direction gives us the PCA-SIFT features.

  2. Generating training codewords: Using a K-means clustering algorithm[2] the calculated PCA-SIFT features were clustered in a certain number of cluster centers. The effect of choice of number of clusters on the recognizer was also studied in addition to using the recommended number of 100 given in the problem.

  3. Feature Vector: Finally a feature vector set for a training images was generated by creating a histogram of the codewords obtained from step2 above. These can now be used by the recognizer to make a decision. A few of the histograms are plotted in the results section below.

- Recognizer

  1. N-Nearest Neighbour Recognizer: A N-Nearest neighbour recognizer was implemented. Herein, PCA-SIFT features of the testing image

were calculated. Then the distance each feature from the center was computed. The following code snippet illustrates my implementation of determining the distance of each feature from the center using the conventional distance formula ;

```
for(int i=0; i<feature.rows; i++)
{
        for(int j=0; j<(kClusters); j++)
        {
                for(int k=0; k<(kComponents); k++)
                {
                        distance[i][j] += pow(feature.at<double>(i,k) -
                        centers.at<float>(j,k), 2.0);
                }
                distance[i][j] = sqrt(distance[i][j]);
        }
}
```

2. Histogram Generation: Once, the distances of features from the centers were calculated, the histogram of the ascending arranged distances was created. This histogram will now be used to used to recognize the object by selecting the image whose training and testing histograms have minimum Euclidean difference.

3. Voting based on nearest neighbours: A voting based decision system was created using the ascendingly sorted array of the minimum differences between the two histograms from step2. A specific number of nearest neighbours was used to make this voting decision and the effect of varying the nearest neighbour on recognizer was also studied. The following code snippet illustrated my implementation;

```
for(int i=0; i<(kNearestNeighbors); i++)
{
        if(sortedDifference.at<int>(i) >= 0 && sortedDifference.at<int>(i) < 20)
                vote[0] += diffBetweenHistograms[sortedDifference.at<int>(i)];

        else if(sortedDifference.at<int>(i) >= 20 && sortedDifference.at<int>(i) < 40)
                vote[1] += diffBetweenHistograms [sortedDifference.at<int>(i)];

        else if(sortedDifference.at<int>(i) >= 40 && sortedDifference.at<int>(i) < 60)
                vote[2] += diffBetweenHistograms [sortedDifference.at<int>(i)];

        else if(sortedDifference.at<int>(i) >= 60 && sortedDifference.at<int>(i) < 80)
                vote[3] += diffBetweenHistograms [sortedDifference.at<int>(i)];

        else
                vote[4] += diffBetweenHistograms [sortedDifference.at<int>(i)];
}
```

4. Decision : Based on the index position of the maximum number of votes in the vote array of 5 elements with index0->4 representing image set of {‘car’, ‘face’, ‘cougar’, ‘pizza’, ‘sunflower’} respectively, I select my recognized object.

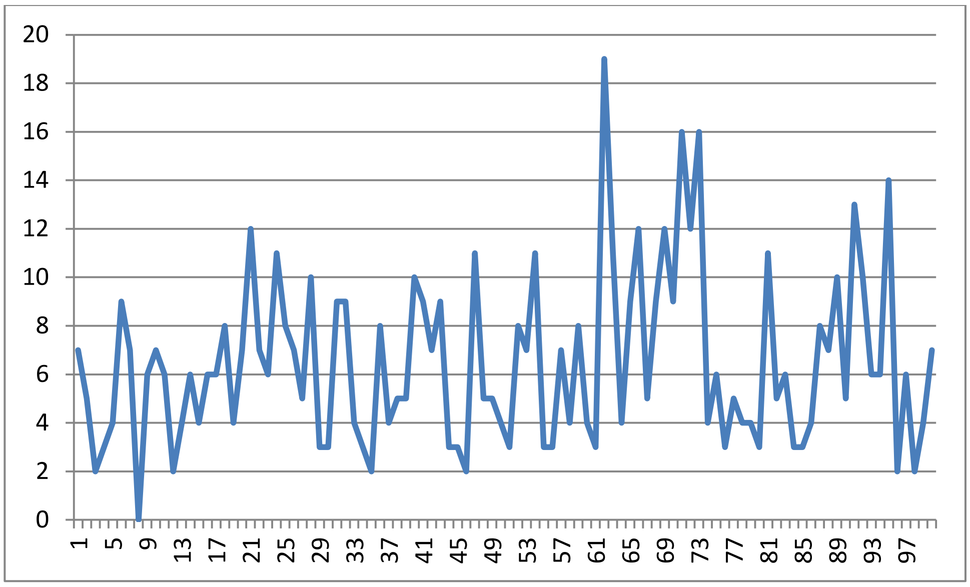


Fig1. Histogram of the feature vectors for the 100 clusters for car image in the training set

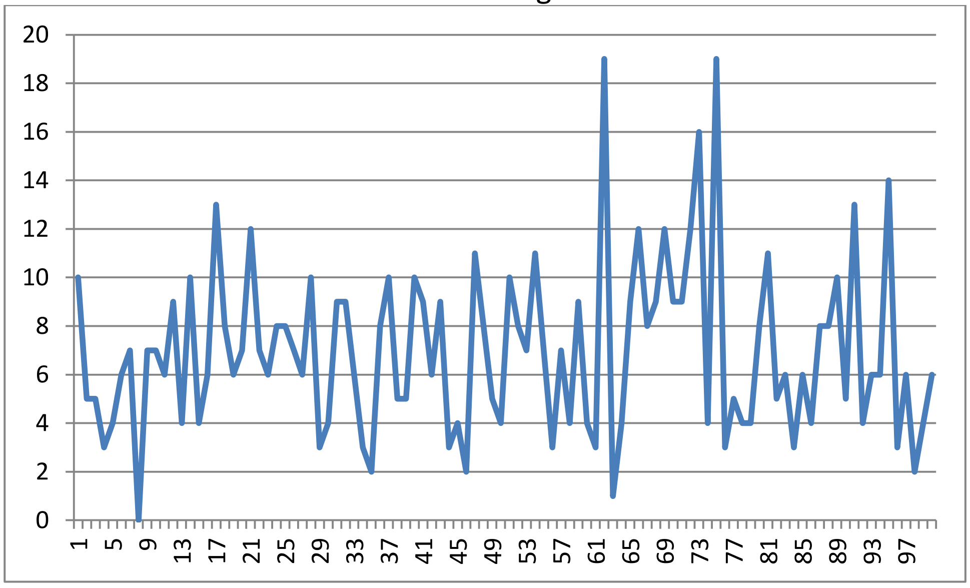


Fig2. Histogram of the feature vectors for the 100 clusters for car image in the testing set.

We can observe that the general trends for the car features are similar and it turns out that all of the testing images for car with the above parameters were identified correctly.

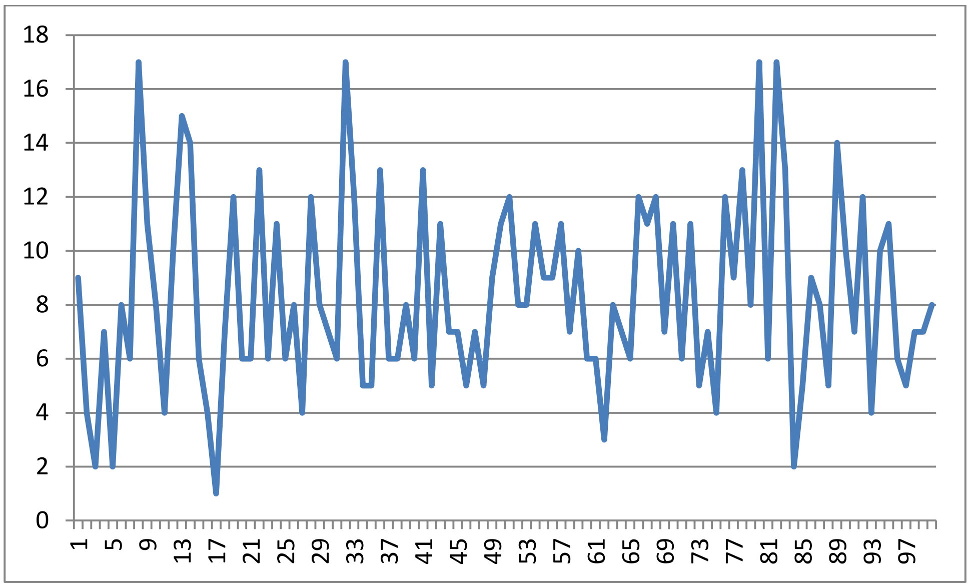


Fig3. Histogram of the feature vectors for the 100 clusters for cougar image in the training set

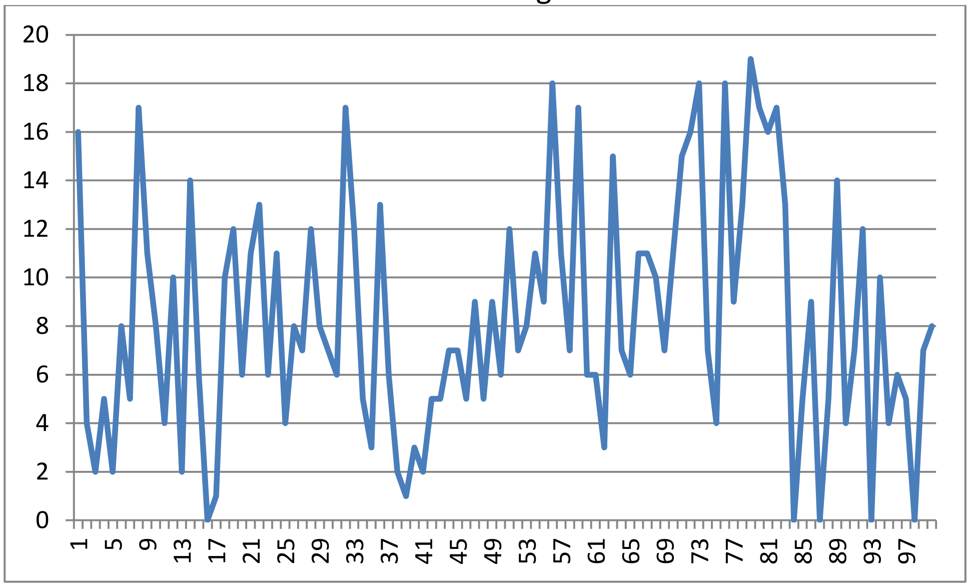
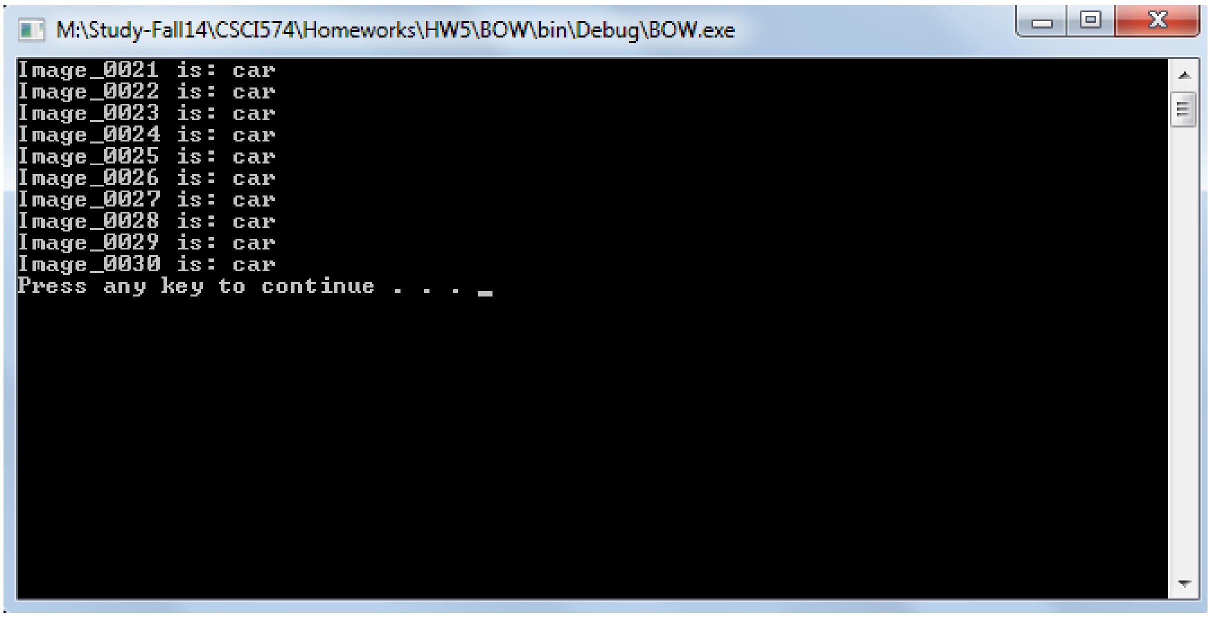


Fig4. Histogram of the feature vectors for the 100 clusters for cougar image in the testing set

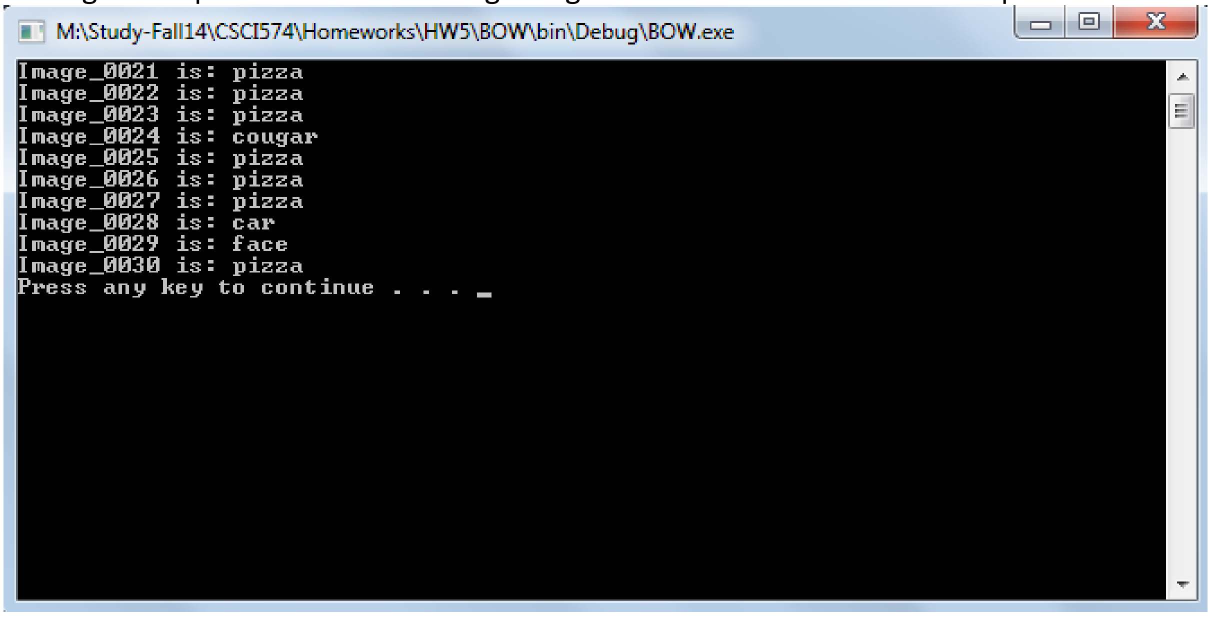
We can observe that the general trends for the cougar features are not very coherent and it turns out that the recognition accuracy for cougar with the above parameters is lesser than the car and there are many mis-classifications of cougar with sunflower and some misclassification with face and pizza as well.

Here is how outputs were displayed on the command prompt;


M:\Study-Fall14\CSCI574\Homeworks\HW5\BOW\bin\Debug\BOW.exe

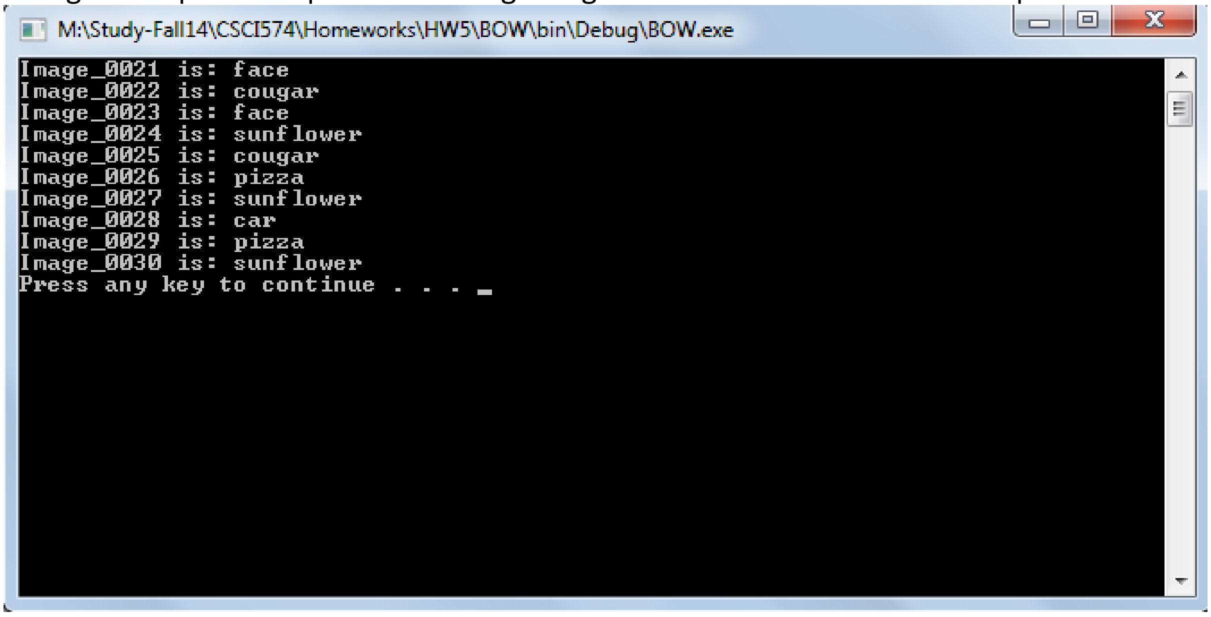
```
Image_0021 is: car
Image_0022 is: car
Image_0023 is: car
Image_0024 is: car
Image_0025 is: car
Image_0026 is: car
Image_0027 is: car
Image_0028 is: car
Image_0029 is: car
Image_0030 is: car
Press any key to continue . . .
```

Fig5. Output for car testing images for the aforementioned parameters


M:\Study-Fall14\CSCI574\Homeworks\HW5\BOW\bin\Debug\BOW.exe

```
Image_0021 is: pizza
Image_0022 is: pizza
Image_0023 is: pizza
Image_0024 is: cougar
Image_0025 is: pizza
Image_0026 is: pizza
Image_0027 is: pizza
Image_0028 is: car
Image_0029 is: face
Image_0030 is: pizza
Press any key to continue . . .
```

Fig6. Output for pizza testing images for the aforementioned parameters


M:\Study-Fall14\CSCI574\Homeworks\HW5\BOW\bin\Debug\BOW.exe

```
Image_0021 is: face
Image_0022 is: cougar
Image_0023 is: face
Image_0024 is: sunflower
Image_0025 is: cougar
Image_0026 is: pizza
Image_0027 is: sunflower
Image_0028 is: car
Image_0029 is: pizza
Image_0030 is: sunflower
Press any key to continue . . .
```

Fig7. Output for sunflower testing images for the aforementioned parameters

Results and Discussion:

The recognizer was run 10 times for different parameter combination.The KNN recognizer worked best for car and face followed by pizza, cougar and sunflower in that order. Face and car were identified correctly with accuracy>90% and in case of car it was 100% in most of the cases.Some mis-classification were observed in case of cougar and face but those were outliers around 15%. Pizza was also consistently identified at around 60-70% accuracy followed by cougar with 40-50% and sunflower recognition was least accurate with 30-40% accuracy. This may be because in many observations, sunflower-face clustering were close to each other so were cougar-sunflower, and face-sunflower and more often than not sunflower and cougar were mis-classified rather than the face. Similar is the case with face-cougar-pizza clustering wherein the colour variation and distribution of those 3 are similar and KMeans would cluster cougar-pizza-sunflower close to face rather than the other way round. This explains the high face recognition accuracy.

Effect of Variation of Parameters on the Recognizer:

Number of PCA components: PCA-based SIFT local descriptors are more distinctive, more robust to image deformations, and more compact than the standard SIFT representation. The main effect of this resulting into faster training and matching as the dimensionality of the problem is reduced. Comparing the results where the clusters are constant at 100 and nearest neighbours are constant at 10 but only principal components are varied from 20 to 30, it is found through the confusion matrix that higher number of principal components gave more recognition accuracy in the case of car (90% -> 100%) and sunflower (30% -> 40%) in this particular case. The cost of this being longer running time and complexity of the recognizer.

Number of K-Means Clusters: The level of detail within the given image decides the optimum level of clusters. An image with uniform components will be unnecessarily be split the uniform object into different clusters even though they belong to the same parent object because the KMeans overclassified due to higher number of clusters.The same gos other way around , the level of detail will be lost if an image with many objectsif KMeans is performed using low amount of clusters. In our case since the car occupies a bigger area within the image and we are not really interested in the background, lesser number of clusters may favour the car image but it also depends on what we have as the background. Face and cougar would be similar in clustering mechanism as the eyes, nose, ears etc. features of face are similar in number even thoughtheyr size and texture may differ. Similiar is the case with pizza and sunflower. Sunflower being hardest as essentially it has only 3 components the center, the petals and stem and ideally they should be clusters as that but since the number of clusters is high, every petal and pixel variation within the center will be clustered differently giving less accuracy.

Number of Nearest Neighbours for Recognizer: Larger value of nearest neighbours avoids misclassification and also avoids unnecessary extraneous or irrelevant features. However, as nearest neighbours were increased, the difference or the boundaries between the different votes became lesser and lesser and again misclassification began to occur as the difference between votes was so less that the recognizer no longer was able to sample those as different votes. This is particularly evident in car and face where as the nearest neighbours increased car image began to degrade in terms of recognition accuracy. So was the case with sunflower and face as well. A more robust method like SVM could also be used in place of KNN in our case.

Confusion matrix and error analysis for 4 such observations are given as follows;

Error Analysis: Following are the confusion matrices and accuracy statistics for four different combinations of input parameters;

- For the input parameters PCA-Components=20 ; K-Means Clusters = 125 and Nearest Neighbours = 15

	Car	Cougar	Face	Pizza	Sunflower
Car	1	0	0	0	0
Cougar	0.4	0.4	0.1	0.1	0
Face	0.3	0	0.7	0	0
Pizza	0.2	0.1	0	0.7	0
Sunflower	0.2	0	0	0.2	0.6

$$Accuracy = \frac{trace\ of\ the\ matrix}{(\sum all\ matrix\ elements)} [3]$$

$$Accuracy = 68\%$$

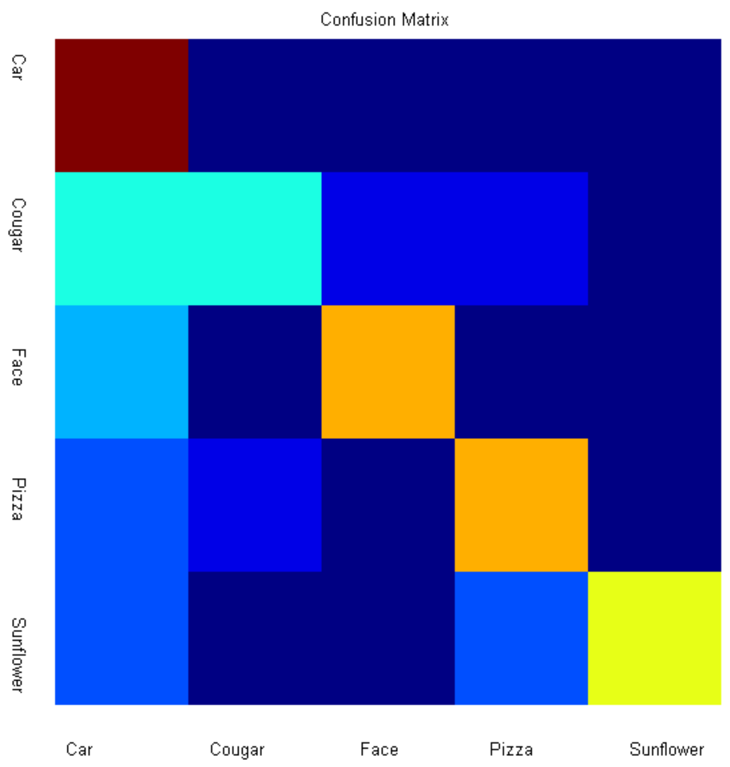


Fig8. Confusion Matrix for PCA-Components=20 ; K-Means Clusters = 125 and Nearest Neighbours = 15.

- For the input parameters PCA-Components=20 ; K-Means Clusters = 100 and Nearest Neighbours = 10.

	Car	Cougar	Face	Pizza	Sunflower
Car	0.9	0.1	0	0	0
Cougar	0.3	0.5	0	0.2	0
Face	0	0	1	0	0
Pizza	0.2	0.1	0	0.7	0
Sunflower	0.1	0.1	0.2	0.2	0.4

$$Accuracy = \frac{trace\ of\ the\ matrix}{(\sum all\ matrix\ elements)}$$

$$Accuracy = 70\%$$

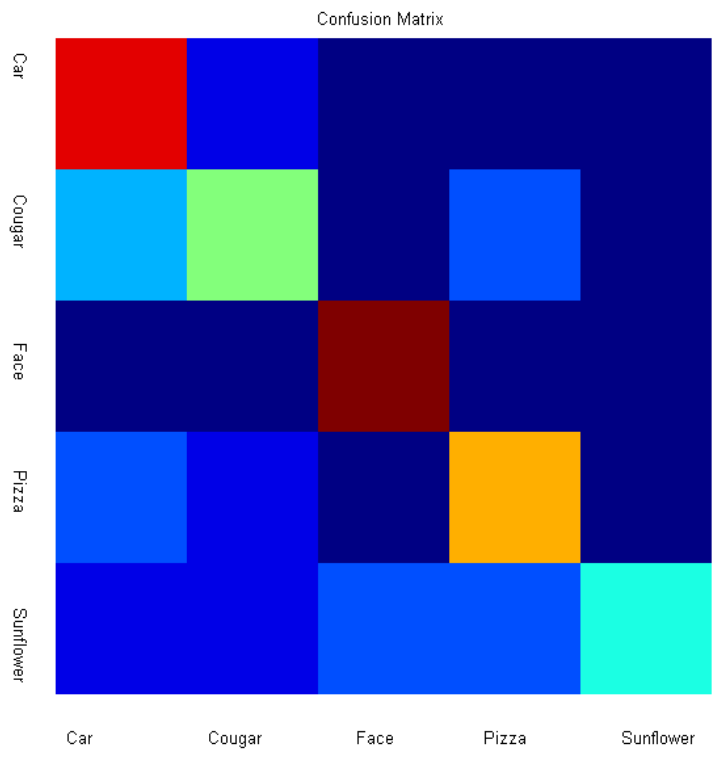


Fig9. Confusion Matrix for PCA-Components=20 ; K-Means Clusters = 100 and Nearest Neighbours = 10.

- For the input parameters PCA-Components=30 ; K-Means Clusters = 100 and Nearest Neighbours = 10.

	Car	Cougar	Face	Pizza	Sunflower
Car	1	0	0	0	0
Cougar	0.2	0.5	0.1	0.2	0
Face	0	0	1	0	0
Pizza	0.2	0.1	0	0.7	0
Sunflower	0.1	0.1	0.2	0.2	0.4

$$Accuracy = \frac{trace\ of\ the\ matrix}{(\sum all\ matrix\ elements)}$$

$$Accuracy = 73.46\%$$

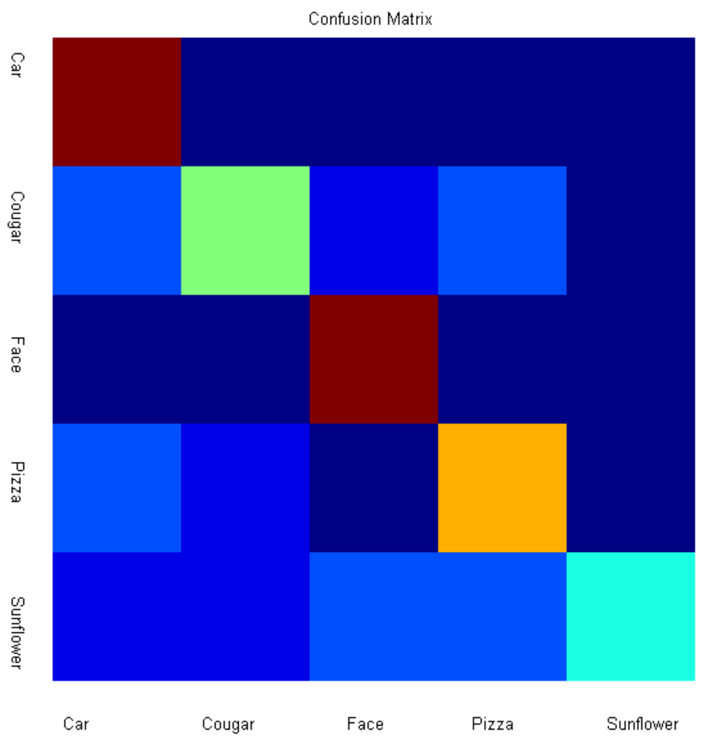


Fig10. Confusion Matrix for PCA-Components=30 ; K-Means Clusters = 100 and Nearest Neighbours = 10.

- For the input parameters PCA-Components=50 ; K-Means Clusters = 150 and Nearest Neighbours = 30.

  The numerical confusion matrix is calculated as;

	Car	Cougar	Face	Pizza	Sunflower
Car	0.9	0.1	0	0	0
Cougar	0.5	0.5	0	0	0
Face	0.4	0	0.6	0	0
Pizza	0.2	0.1	0.1	0.6	0
Sunflower	0.4	0.2	0.1	0	0.3

$$Accuracy = \frac{trace\ of\ the\ matrix}{(\sum all\ matrix\ elements)}$$
$$Accuracy = 58\%$$

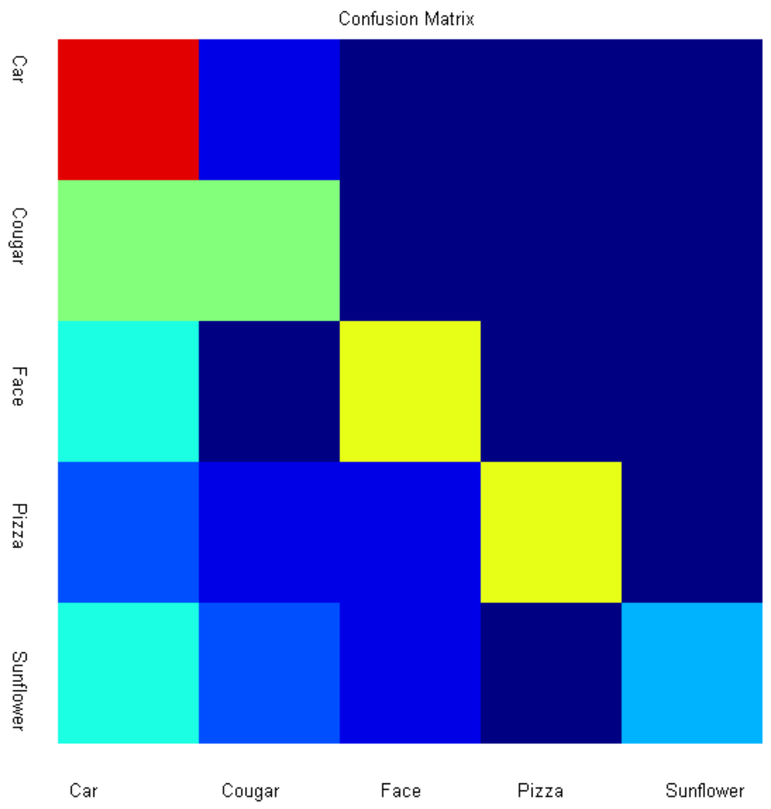


Fig11. Confusion Matrix for PCA-Components=50 ; K-Means Clusters = 150 and Nearest Neighbours = 30.

Similarly accuracy for all the 10 variations in parameters was calculated and average accuracy for the recognizer turned out be 71.44%

Source Code:

```
/**
 * CSCI 574 : Computer Vision
 * Homework # 5 : Object Recognition using Bag of Words and K-Nearest Neighbours
 * Author: Manan Vyas
 * USCID:  7483-8632-00
 * Email:  mvyas@usc.edu
 */

// -------- LocalFeatures.h -----------

// C/C++ specific
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <string>
#include <fstream>
#include <vector>

// OpenCV specific
#include <opencv2/core/core.hpp>
#include <opencv2/core/core_c.h>
#include <opencv2/nonfree/nonfree.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/imgproc/types_c.h>
#include <opencv2/legacy/legacy.hpp>

#ifndef __LOCALFEATURES__
#define __LOCALFEATURES__

// Variables and Data Structures needed for both training and recognition
int const kNumofTrainingImagesSet = 5;
int const kNumofImagesInEachTrainingSet = 20;
int trainingPoints[100] = {0};
Mat SIFT_features;
Mat descriptor;
Mat eigenVectors;
Mat principalDirections;
Mat codeWords;
Mat labels, centers;
Mat bestLabels;
int trainingPoints[100];

// Mechanism to read all the training/Testing datasets
Mat myIO(int kNumofTrainingImagesSet);

// Compute SIFT features
cv::Mat SIFT_features(cv::Mat img, int retVal);

// PCA
cv::Mat principalComponentAnalysis (cv::Mat SIFT_features);

// Generate CodeWords for PCA-SIFT
cv::Mat generateCodewords (cv::Mat eigenVectors, cv::Mat SIFT_features);

// Kmeans
cv::Mat K_Means (cv::Mat codeWords, int kClusters);

// Generate Feature Vector
void featureVector (Mat codeWords, int const kClusters, int siftNum[], int const kLength)

#endif
```

```
/**
 * CSCI 574 : Computer Vision
 * Homework # 5 : Object Recognition using Bag of Words and K-Nearest Neighbours
 * Author: Manan Vyas
 * USCID:  7483-8632-00
 * Email:  mvyas@usc.edu
 */

// -------- LocalFeatures.cpp -----------

// header
#include "LocalFeatures.h"

// C/C++ specific
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <string>
#include <fstream>
#include <vector>

// OpenCV specific
#include <opencv2/core/core.hpp>
#include <opencv2/core/core_c.h>
#include <opencv2/nonfree/nonfree.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/imgproc/types_c.h>
#include <opencv2/legacy/legacy.hpp>

using namespace std;
using namespace cv;


//***
// Method to calculate SIFT_PCA features
// Input: Image whose features have to be calculated, return type
//***
Mat SIFT_features(Mat img,int retVal)
{
    SIFT siftFeatureCompute;
    vector<KeyPoint> SIFTkeypoints;
    Mat SIFT_features;
    Mat descriptor;

       // Reference:

    siftFeatureCompute.operator()(img, noArray(), SIFTkeypoints, descriptor, false);
    SIFT_features.push_back(descriptor);

    if (retVal == 1)
        return descriptor;
    else
        return SIFT_features;
}
```

```
// ***
// Method to perform PCA using the eigen vectors and reduce the dimensionaltiy of the
problem
// ***
Mat principalComponentAnalysis (Mat SIFT_features)
{
    Mat covarianceMatrix, mean;
    Mat eigenValues,eigenVectors;

    // Calculate covariance matrix for PCA
    // Reference : OpenCV docs
    //URL<http://docs.opencv.org/modules/core/doc/operations_on_arrays.html?
    //    highlight=calccovarmatrix#void%20calcCovarMatrix
    //
(const%20Mat*%20samples,%20int%20nsamples,%20Mat&%20covar,%20Mat&%20mean,%20int%20flag
s,%20int%20ctype)>
    calcCovarMatrix(SIFT_features, covarianceMatrix, mean, CV_COVAR_NORMAL |
CV_COVAR_ROWS, CV_64F);

    // Calculate Eigen decomposition to reduce the dimensionalty
    // Reference OpenCV docs :
    // <URL><http://docs.opencv.org/modules/core/doc/operations_on_arrays.html?
    // highlight=eigen#bool%20eigen
    //
(InputArray%20src,%20OutputArray%20eigenvalues,%20OutputArray%20eigenvectors,%20int%20
lowindex,int%20highindex)>
    eigen(covarianceMatrix, eigenValues, eigenVectors);
    return eigenVectors;
}

// ***
//Funtion to generate PCA-SIFT codewords.
//Inputs: EigenVectors,SIFT-Features, Number of Principal Directions
// ***
Mat generateCodewords (Mat eigenVectors, Mat SIFT_features, int const
kPrincipalDirections)
{
    Mat principalDirections,pDt;
    Mat codeWords;
    // 20 components considered as advised in the homeword question ...
    for(unsigned int i=0; i<(kPrincipalDirections); i++)
        principalDirections.push_back(eigenVectors.row(i));

    // .T() ->Transpose to make matrix multiplication possible !
    pDt = principalDirections.t();
    Mat temp;
    (SIFT_features.convertTo(temp, CV_64F));
    // Generate codeWords...
    codeWords = (temp)*(pDt);
    return codeWords;
}

// ***
// K-Means Clustering on PCA-SIFT codeWords genetared
// kClsters around 100-120 range is good
// ***
Mat K_Means (Mat codeWords, int kClusters)
{
    Mat labels, centers;
    codeWords.convertTo(codeWords, CV_32F);

    // K-Means .. with 10 attempts
    kmeans(codeWords, kClusters, labels, TermCriteria(CV_TERMCRIT_EPS, 100, 0.48), 10,
KMEANS_PP_CENTERS, centers );

    return labels;
}
```

```
// ***
// Object Feature vector generation method
// Generates a histogram of codewords for each image to obtain its unique feature
vector
// ***
void featureVector (Mat codeWords, int const kClusters, int siftNum[], int const
kLength)
{
       Mat bestLabels;
       Mat labels, centers;
    codeWords.convertTo(codeWords, CV_32F);

    // K-Means .. with 10 attempts
    kmeans(codeWords, kClusters, labels, TermCriteria(CV_TERMCRIT_EPS, 100, 0.48), 10,
KMEANS_PP_CENTERS, centers );
       int trainingFeatures[(kLength)][kClusters] = {0};
       int step = 0;
       for(int i=0; i<100; i++)
       {
              for(int j=0; j<siftNum[i]; j++)
              {
                     if(i==0)
                            trainingFeatures[i][bestLabels.at<int>(j)]++;
                     else
                            trainingFeatures[i][bestLabels.at<int>(j + step)]++;
                     //cout<<bestLabels.at<int>(j);
              }
              step = step + siftNum[i];
       }
}

// ***
// Function to read and analyse the input images ...
// This calculates and analysis the input images WRT training data
// ***
Mat myIO(int kNumofTrainingImagesSet)
{
    int countVar = 0;
    String trainingSetImageName;
    Mat descriptor;

    // Points of interest for training the system/recognizer
    int trainingPoints[100] = {0};

    while (countVar != kNumofTrainingImagesSet)
    {
        // Switch case to select the appropriate folder structure
        switch(countVar)
        {
            case 0 :
                trainingSetImageName = "M:/Study-
Fall14/CSCI574/Homeworks/HW5/dataset/images/training/car/";
                break;
            case 1 :
                trainingSetImageName = "M:/Study-
Fall14/CSCI574/Homeworks/HW5/dataset/images/training/cougar/";
                break;
            case 2 :
                trainingSetImageName = "M:/Study-
Fall14/CSCI574/Homeworks/HW5/dataset/images/training/face/";
                break;
            case 3 :
                trainingSetImageName = "M:/Study-
Fall14/CSCI574/Homeworks/HW5/dataset/images/training/pizza/";
                break;
              case 4 :
                     trainingSetImageName = "M:/Study-
                     Fall14/CSCI574/Homeworks/HW5/dataset/images/training/sunflower/";
              break;
        }
        // Read each image set within the training set
        for (unsigned int i=1 ; i<=20 ; i++)
        {
            Mat img; // To store actual individual images that areour training data
            //stringstream
            ostringstream oss;
            // final individual image filename
            string fileName;
            // 0001->0020 are the image names
            if (i<10) // 0001->0009
            {
                oss << i;
                fileName = trainingSetImageName+"image_000"+oss.str()+".jpg";
            }
            else // 0010->0020
            {
                oss << i;
                fileName = "image_00"+oss.str()+".jpg";
            }
            //*completed read all 20 images from one dataset of images
            // Read indiavial images into Mats
            // Note : Car images are grey images in the given data set .. hence the
if..else structure .. not really needed

            if (countVar == 0) // Read grey Images
            {
                img = imread(fileName,0);

            }
            else // color image
            {
                cvtColor((imread(fileName,1)),img,COLOR_BGR2GRAY);
            }

            // Calculate SIFT_PCA Features and store it in the data structure ...
            descriptor = SIFT_features(img, 1);
            trainingPoints[countVar*20 + i - 1] = descriptors.rows;

        }// end of for loop
        // next 'SET' of training Images
        countVar = countVar+1;
    }//end of loop
}//end of myIO function ...
```

```cpp
/**
 * CSCI 574 : Computer Vision
 * Homework # 5 : Object Recognition using Bag of Words and K-Nearest Neighbours
 * Author: Manan Vyas
 * USCID:  7483-8632-00
 * Email:  mvyas@usc.edu
 */

// // -------- BOW.cpp -----------

#include "LocalFeatures.h"

// C/C++ specific
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <string>
#include <fstream>
#include <vector>

// OpenCV specific
#include <opencv2/core/core.hpp>
#include <opencv2/core/core_c.h>
#include <opencv2/nonfree/nonfree.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/imgproc/types_c.h>
#include <opencv2/legacy/legacy.hpp>

using namespace cv;
using namespace std;

// Imporatant Parameters affecting recognition accuracy and performance
int const kClusters = 100;
int const kPrincipalComponents = 30;
int const kNearestNeighours = 10;

// Variables and Data Structures needed for both training and recognition
int const kNumofTrainingImagesSet = 5;
int const kNumofImagesInEachTrainingSet = 20;
int trainingPoints[100] = {0};
// Feature Vectors
int trainingFeatures[100][(kClusters)] = {0};

Mat SIFT_features;
Mat descriptor;
Mat eigenVectors;
Mat principalDirections;
Mat codeWords;
Mat labels, centers;
```

```
// Function that call all the necessary functions to compute and store the training
features
// Stores the requisite information and trained features into the variables declared
above as
// they will be needed for recoginiton purposes as well
void train()
{
       int const kLength = 100;
       // Raed all the Images for training
       // Internally calls SIFT-Features to generate the SIFT features for all the
images as well
       // Ref: LocalFeatures.h ; and LocalFeatures.cpp and SIFT_features()
       // trainingPoints/decsriptors available implicitly via global variable if
required
       SIFT_Features = myIO(kNumofTrainingImagesSet);
       // perform pca
       eigenVectors = principalComponentAnalysis(SIFT_Features);
       // generate codeWords
       pcaSift = generateCodewords(eigenVectors,SIFT_features,kPrincipalComponents);
       // KMeans Clustering
       labels = K_Means(pcaSift,kClusters);
       // generate feature vectors
       // trainingPoints implicitly available .. they are computer by myIO function

       featureVector(pcaSift,kClusters,trainingPoints, (kLength));
       int trainingFeatures[(kLength)][kClusters] = {0};
       int Idx = 0;
       for(int i=0; i<(kLength); i++)
       {
              for(int j=0; j<trainingPoints[i]; j++)
              {
                     trainingFeatures[i][labels.at<int>(j + step)]++;
                     //cout<<bestLabels.at<int>(j);
              }
              Idx += trainingPoints[i];
       }
       // int trainingFeatures[100][(kClusters)] contains all the feature vectors ..
implicitly available through global variable
       cout<<"End of training .... "<<endl;
}

void recognize()
{
       // Object  Recognition
       // Interate through all the testing images
       for (int ii=21; ii<=30; ii++) // Main loop all computations done here
       // Iterations over 21->30 because images are named as Image_0021->Image_0030
       {
              // Read the image
              ostringstream oss;
              string file;
        oss << ii;
        file = "M:/Study-
Fall14/CSCI574/Homeworks/HW5/dataset/images/testing/pizza/image_00" + oss.str() +
".jpg";

              //Mat testImg = imread("test_car_10.jpg", 0);

              Mat testImg = imread(file, 1);
              Mat grayTestImg;
              cvtColor(testImg, grayTestImg, CV_BGR2GRAY);

              // Compute 128-dimension SIFT features for each image
              SIFT sift;
              vector<KeyPoint> keypoints;
              Mat descriptor;
```

```
		sift.operator()(grayTestImg, noArray(), keypoints, descriptors, false);
		Mat feature;
		Mat descriptors64f;
		descriptors.convertTo(descriptors64f, CV_64F);
		// Dot Product between SIFT descriptors and the primary directions to get
PCA-SIFT features
		feature = descriptors64f * priDir_t;

		// Distance computation between the feature vector and the label center of
the cluster
		double **distance;
		distance = new double * [feature.rows];

		// Initialize the distance data structure
		for(int i=0; i<feature.rows; i++)
		{
			distance[i] = new double [(kClusters)];
			for (int j=0; j<(kClusters); j++)
				distance[i][j] = 0;
		}

		// Compute the distance of each feature vector in the testing image from
the labelCenters
		// i: each feature; j: to each center
		for(int i=0; i<feature.rows; i++)
		{
			for(int j=0; j<(kClusters); j++)
			{
				for(int k=0; k<(kPrincipalComponents); k++)
				{
					// Euclidean Distance between feature vector and label
center
					// d = sqrt(trainingFeatures of all elements ((feature-
center)^2))
					distance[i][j] += pow(feature.at<double>(i,k) -
centers.at<float>(j,k), 2.0);
				}
				distance[i][j] = sqrt(distance[i][j]);
			}
		}

		// Histogram of the feature vectors of the testing image
		int histogram[kClusters] = {0};
		// holds the minimum distances between the saved template histogram and
currently calculated histogram
		vector<int> minimumDistance;

		for(int i=0; i<feature.rows; i++)
		{
			int temp=0;
			for(int j=1; j<(kClusters); j++)
			{
				// Choose minima
				if(distance[i][temp] > distance[i][j])
					temp = j;
			}
			// store in the vector in ascending order
			minimumDistance.push_back(temp);
		}
		// Create the histogram of the mimimum distances
		for(int i=0; i<feature.rows; i++)
		{
			histogram[minimumDistance[i]]++;
		}

		// Calculate the difference btween the features of training and testing 
images
		// by calculate the Euclidean distance betweein the constituents of the
		// histograms of feature vectors of the training images and the 
constituents of
		// the histogram of the feature vectore of the testing image.
		double difference[100] = {0};
		for(int i=0; i<100; i++)
		{
			for(int j=0; j<(kClusters); j++)
			{
				// Difference = sqrt(trainingFeatures over all 
elements((current-template)^2))
				difference[i] += (histogram[j] - trainingFeatures[i][j]) * 
(histogram[j] - trainingFeatures[i][j]);
			}
			difference[i] = sqrt(difference[i]);
		}

		Mat sortedDifferences;
		Mat src(1,100,CV_64F,&difference);
		// Sort the differenceerence array for the vote
		sortIdx(src,sortedDifferences,CV_SORT_EVERY_ROW | CV_SORT_ASCENDING );

		// Once we have the selected histogram that has the mimimum difference 
between the histogram
		// of the training feature vector and the histogram containing feature 
vector of the testing image
		// Generate a Voting based decision
		// Array of 5 index corresponds to the 5 sets of testing images
		double vote[5] = {0};
		for(int i=0; i<(kNearestNeighbors); i++)
		{
			if(sortedDifferences.at<int>(i) >= 0 && 
sortedDifferences.at<int>(i) < 20) // For Car
				vote[0] += difference[sortedDifferences.at<int>(i)];
			else if(sortedDifferences.at<int>(i) >= 20 && 
sortedDifferences.at<int>(i) < 40) // For Cougar
				vote[1] += difference[sortedDifferences.at<int>(i)];
			else if(sortedDifferences.at<int>(i) >= 40 && 
sortedDifferences.at<int>(i) < 60) // For Face
				vote[2] += difference[sortedDifferences.at<int>(i)];
			else if(sortedDifferences.at<int>(i) >= 60 && 
sortedDifferences.at<int>(i) < 80) // For Pizza
				vote[3] += difference[sortedDifferences.at<int>(i)];
			else										
									// For Sunflower
				vote[4] += difference[sortedDifferences.at<int>(i)];
		}
```

```
              int max=0;
              // Find the index where we get maximum votes
              // The one with the maximum votes is decided to be the recognized image
              for(int i=0; i<5; i++)
              {
                     // Calculate the maxima
                     if(vote[max] < vote[i])
                            max=i;
              }
              // Message on sceen to display our decision for the recognizer
              // Regarding the input image belongs to what category
              switch(max)
              {
                     case 0:
                            cout<<"Image_00"<<ii<<" is: car"<<endl;
                            break;
                     case 1:
                            cout<<"Image_00"<<ii<<" is: cougar"<<endl;
                            break;
                     case 2:
                            cout<<"Image_00"<<ii<<" is: face"<<endl;
                            break;
                     case 3:
                            cout<<"Image_00"<<ii<<" is: pizza"<<endl;
                            break;
                     case 4:
                            cout<<"Image_00"<<ii<<" is: sunflower"<<endl;
                            break;
              }
              // Deallocate Memory
              delete[] distance;
       }
       cout<<"Object Recognized ...." <<endl;
}

int main(int argc, char* argv[])
{
       train();
       recognize();
       system("pause");
       return 0;
}
```

References:

[1]<URL>http://docs.opencv.org/modules/nonfree/doc/feature_detection.html

[2]<URL>< http://docs.opencv.org/modules/core/doc/clustering.html>

[3]<URL><www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html>