CSCI 574 HW 5
Ahmet Can Ozbek

## Introduction

In this assignment, we are asked to implement "bag of features" method, for object recognition. We have five object classes, the object classes we have are:

- Car Side
- Butterfly
- Face
- Watch
- Water Lilly

In the next section, we are going to explain the procedure step by step.

## Methodology

Here, in this section we are going to give the outline the procedure step by step and explain each of them. Steps involved are:

1) SIFT feature extraction of all training images
2) Apply PCA to all extracted data features
3) Apply k-means clustering to obtain the codewords
4) Obtain the histogram of codewords of training images
5) Train the system by "k nearest neighbor" method by training data
6) Get the classification results for both training and testing data

### 1) SIFT feature extraction of all training images

At this step, we are just using SIFT feature extraction to get 128 dimensional vectors for each keypoint in the images.

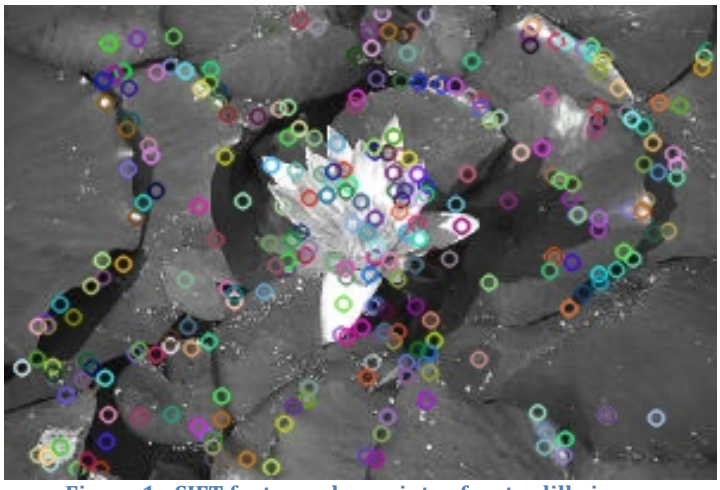


Figure 1 - SIFT features, keypoints of water lilly image

## 2) Applying PCA to all extracted SIFT features

After applying SIFT feature extraction to each image, we get 128 dimensional vectors for each keypoint in each image in the training dataset. We are asked to apply PCA to reduce dimensionality of 128 to some lower number. In this example I chose to reduce the dimensionality to 20.

To apply PCA, I merged all keypoint feature vectors of all training images into one single matrix and then applied PCA.

## 3) Apply k-means clustering to obtain the keywords

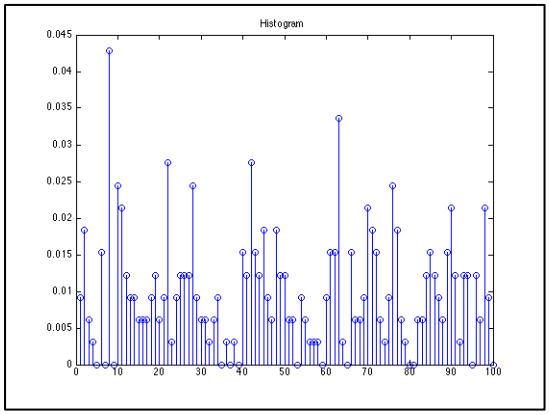
After reducing the dimensionality from 128 to 20, we apply k-means clustering to obtain the codeword. Each cluster will be the visual codewords to represent images as a histogram, therefore the parameter k in k-means clustering will determine the number of visual codewords, hence the number of bins in the histogram. In this example we choose k to be 100.

## 4) Obtain the histogram of codewords of training images

After getting the visual codewords, we need to get the histogram of each image to perform the classification. We need to get the visual codeword histogram of each image.

Each PCA-SIFT feature of the image should be quantized to the closest codeword. After this quantization of each PCA-SIFT of each image, we will count the number of occurance of each codeword, this will give us a histogram for each image. In our example, as there are 100 clusters in k-means clustering algorithm, our histograms will have 100 bins, hence each image will have 100 dimensional vector representing its visual codeword content.

After getting the histogram of a given image, we are normalizing the histogram so that each element will have a number between 0 and 1.
Below, we will see the histogram generated by this algorithm for our example water lilly image.



## 5) Train the system by "k nearest neighbor" method by training data

In our example, we took the number of clusters in k-means clustering to be 100, therefore our histogram features for each image will have dimension 100. By using these features in the training data, we train our classifier k nearest neighbor (kNN).

## 6) Get the classification results for both training and testing data

After training our system, we can get the error rate for each class in training and testing data.

### Classification Results

Below, we have the error rate results for training and testing data according to the three parameters: PCA dimension, number of clusters (k), k in kNN classifier.

| PCA | k Clusters | k in KNN | Training Error(%) | Testing Error(%) |
|---|---|---|---|---|
| 15 | 50 | 3 | 19 | 30 |
| 15 | 50 | 10 | 34 | 34 |
| 15 | 50 | 20 | 40 | 44 |
| 15 | 100 | 3 | 19 | 24 |
| 15 | 100 | 10 | 37 | 42 |
| 15 | 100 | 20 | 36 | 42 |
| 15 | 150 | 3 | 20 | 28 |
| 15 | 150 | 10 | 27 | 34 |
| 15 | 150 | 20 | 30 | 44 |
| 20 | 50 | 3 | 18 | 32 |
| 20 | 50 | 10 | 30 | 38 |
| 20 | 50 | 20 | 46 | 52 |
| 20 | 100 | 3 | 21 | 22 |
| 20 | 100 | 10 | 34 | 40 |
| 20 | 100 | 20 | 42 | 50 |
| 20 | 150 | 3 | 18 | 28 |
| 20 | 150 | 10 | 33 | 40 |
| 20 | 150 | 20 | 39 | 46 |
| 25 | 50 | 3 | 18 | 38 |
| 25 | 50 | 10 | 32 | 46 |
| 25 | 50 | 20 | 43 | 54 |
| 25 | 100 | 3 | 20 | 32 |
| 25 | 100 | 10 | 34 | 50 |
| 25 | 100 | 20 | 39 | 50 |
| 25 | 150 | 3 | 21 | 34 |
| 25 | 150 | 10 | 32 | 44 |
| 25 | 150 | 20 | 36 | 46 |

From the above results, we significantly observe that the "k" parameter in our kNN classifier determines the performance and we observe that if "k" is low, we get better performance. This might be due to the fact that the number of samples in our training data is not that high, therefore decision on lower number of neighbors is giving us better result.

Here, I will also show the confusion matrix for the parameters that gave a good result.

In the confusion matrices below, parameters are

- PCA dimension = 20
- Number of Clusters (k) = 100
- k in kNN classifier = 3

Training Data (21% error rate):

| | Car Side | Butterfly | Face | Watch | Water Lilly |
|---|---|---|---|---|---|
| Car Side | 20 | 0 | 0 | 0 | 0 |
| Butterfly | 3 | 15 | 1 | 0 | 1 |
| Face | 1 | 0 | 19 | 0 | 0 |
| Watch | 5 | 4 | 3 | 8 | 0 |
| Water Lilly | 2 | 1 | 0 | 0 | 17 |

Testing Data (22% error rate):

| | Car Side | Butterfly | Face | Watch | Water Lilly |
|---|---|---|---|---|---|
| Car Side | 10 | 0 | 0 | 0 | 0 |
| Butterfly | 0 | 8 | 1 | 0 | 1 |
| Face | 2 | 0 | 8 | 0 | 0 |
| Watch | 1 | 1 | 3 | 5 | 0 |
| Water Lilly | 2 | 0 | 0 | 0 | 8 |

## OpenCV C++ Code:

main.cpp

```
//CSCI574 HW5
//Ahmet Can Ozbek
//8401860152
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2/ml/ml.hpp>
#include <vector>
#include "auxFunctions.h"

using namespace std;
using namespace cv;

#define N_OF_CLASSES 5
#define N_OF_TRAINING_IMAGES 100
#define N_OF_TEST_IMAGES 50


int main(int argc, const char * argv[]) {

    /*
    label 0 --> car_side
    label 1 --> butterfly
    label 2 --> faces
    label 3 --> watch
    label 4 --> water_lilly
    */

    //1)Read the images
    //Reading Training Images
    Mat trainImages[N_OF_TRAINING_IMAGES];
    aux::readTrainImages(trainImages, N_OF_TRAINING_IMAGES);
    //Reading Test Images
    Mat testImages[N_OF_TEST_IMAGES];
    aux::readTestImages(testImages, N_OF_TEST_IMAGES);

    //Construct true labels for training and testing images
    Mat trueTrainLabels;
    Mat trueTestLabels;
    aux::getTrueTrainingLabels(trueTrainLabels, N_OF_TRAINING_IMAGES);
    aux::getTrueTestLabels(trueTestLabels, N_OF_TEST_IMAGES);

    int pcaDimension = 25;
    int numberOfClusters = 150;
    int kNeighbors = 20;

    int pcaDimArray[3] = {15,20,25};
    int kArray[3] = {50,100,150};
    int kNNArray[3] = {3,10,20};


    //2)
    PCA pcaSift;
    SIFT siftExtractor;
    Mat descriptorsOfTrainImages[N_OF_TRAINING_IMAGES]; //SIFT descriptor matrix for each
training image


    //Get codewords
    Mat codewords = aux::getWords(trainImages, descriptorsOfTrainImages,
                                  pcaSift, siftExtractor, pcaDimension,
numberOfClusters); //Returns the cluster means (codewords)
```

```
    //Get the histograms of training images
    Mat trainingHistogramData;
    for(int i=0; i<N_OF_TRAINING_IMAGES; i++){
        trainingHistogramData.push_back(aux::getHistogram(trainImages[i], codewords,
siftExtractor, pcaSift));
    }

    //Get the histograms of testing images
    Mat testingHistogramData;
    for(int i=0; i<N_OF_TEST_IMAGES; i++){
        testingHistogramData.push_back(aux::getHistogram(testImages[i], codewords,
siftExtractor, pcaSift));
    }
    trainingHistogramData.convertTo(trainingHistogramData, CV_32F);
    testingHistogramData.convertTo(testingHistogramData, CV_32F);


//    //FOR REPORT START
//    cout << "Histogram: \n" << trainingHistogramData.row(trainingHistogramData.rows -
5) << endl;
//    waitKey();
//    //FOR REPORT END

    //Classification
    //KNN

    cout << "KNN Classification with parameters: " << endl;
    cout << "*PCA dimension: " << pcaDimension << endl;
    cout << "*k(Number Of Clusters in k-means): " << numberOfClusters << endl;
    cout << "*k nearest neighbors in kNN classification: " << kNeighbors << endl;

    //Train kNN classifier
    CvKNearest knnClassifier;
    knnClassifier.train(trainingHistogramData, trueTrainLabels);


    //Results
    cout << "Results: " << endl;
    //Error on training data
    Mat predictedTrainingLabelsKNN;
    knnClassifier.find_nearest(trainingHistogramData, kNeighbors,
&predictedTrainingLabelsKNN);
    predictedTrainingLabelsKNN.convertTo(predictedTrainingLabelsKNN, CV_32S);
    cout << "-Training Error Rate: " << aux::getErrorRate(trueTrainLabels,
predictedTrainingLabelsKNN) << endl;

    //Error on testing data
    Mat predictedTestingLabelsKNN;
    knnClassifier.find_nearest(testingHistogramData, kNeighbors,
&predictedTestingLabelsKNN);
    predictedTestingLabelsKNN.convertTo(predictedTestingLabelsKNN, CV_32S);
    cout << "-Testing Error Rate: " << aux::getErrorRate(trueTestLabels,
predictedTestingLabelsKNN) << endl;

    //Training Confusion Matrix
    Mat confusionMatrixTraining;
    confusionMatrixTraining = aux::getConfusionMatrix(trueTrainLabels,
predictedTrainingLabelsKNN, N_OF_CLASSES);
    cout << "-Training Confusion Matrix: \n" << confusionMatrixTraining << endl;

    //Test Confusion Matrix
    Mat confusionMatrixTest;
    confusionMatrixTest = aux::getConfusionMatrix(trueTestLabels,
predictedTestingLabelsKNN, N_OF_CLASSES);
    cout << "-Testing Confusion Matrix: \n" << confusionMatrixTest << endl;


    cout << "End." << endl;
    return 0;
}
```

auxFunctions.cpp

```
//CSCI574 HW5
//Ahmet Can Ozbek
//8401860152

#include "auxFunctions.h"
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/nonfree/features2d.hpp>
#include <vector>
#include <iostream>
#include <string>

#define N_OF_CLASSES 5
#define N_OF_TRAINING_IMAGES 100
#define N_OF_TRAINING_IMAGES_PERCLASS 20
#define N_OF_TEST_IMAGES_PERCLASS 10
#define N_OF_TEST_IMAGES 50

using namespace cv;
using namespace std;

void aux::readTrainImages(Mat* inputMatArray, int arraySize){

    String fileLocation = "/Users/ahmetcanozbek/Desktop/CSCI574/Homeworks/hw5/images/";
    String folderName = "/train/";
    String fileExtension = ".jpg";
    String fullPathFileName;

    for(int classIndex=0; classIndex<N_OF_CLASSES; classIndex++){
        for(int i=0; i<N_OF_TRAINING_IMAGES_PERCLASS; i++){
            //construct full path filename
            fullPathFileName = fileLocation + getClassName(classIndex) + folderName +
to_string(i+1) + fileExtension;
            inputMatArray[classIndex*N_OF_TRAINING_IMAGES_PERCLASS + i] =
imread(fullPathFileName,0);
        }
    }
}

void aux::readTestImages(Mat* inputMatArray, int arraySize){

    String fileLocation = "/Users/ahmetcanozbek/Desktop/CSCI574/Homeworks/hw5/images/";
    String folderName = "/test/";
    String fileExtension = ".jpg";
    String fullPathFileName;

    for(int classIndex=0; classIndex<N_OF_CLASSES; classIndex++){
        for(int i=0; i<N_OF_TEST_IMAGES_PERCLASS; i++){
            //construct full path filename
            fullPathFileName = fileLocation + getClassName(classIndex) + folderName +
to_string(i+1) + fileExtension;
            inputMatArray[classIndex*N_OF_TEST_IMAGES_PERCLASS + i] =
imread(fullPathFileName,0);
        }
    }
}

void aux::getTrueTrainingLabels(Mat& trueTrainingLabels, int size){

    trueTrainingLabels = Mat::zeros(N_OF_TRAINING_IMAGES, 1, CV_32S);

    for(int classNumber=0; classNumber<N_OF_CLASSES; classNumber++){
        for(int i=0; i<N_OF_TRAINING_IMAGES_PERCLASS; i++){
            trueTrainingLabels.at<int>(classNumber*N_OF_TRAINING_IMAGES_PERCLASS+i,0) =
classNumber;
        }
    }
```

```
}

void aux::getTrueTestLabels(Mat& trueTestLabels, int size){

    trueTestLabels = Mat::zeros(N_OF_TEST_IMAGES, 1, CV_32S);

    for(int classNumber=0; classNumber<N_OF_CLASSES; classNumber++){
        for(int i=0; i<N_OF_TEST_IMAGES_PERCLASS; i++){
            trueTestLabels.at<int>(classNumber*N_OF_TEST_IMAGES_PERCLASS+i,0) =
classNumber;
        }
    }
}


Mat aux::getWords(Mat* trainImages, Mat* descriptorsOfTrainImages, PCA& pcaSIFT, SIFT&
siftExtractor, int pcaDimension, int numberOfClusters){

    //SIFT feature extraction
    //Get descriptors of training images
    vector<KeyPoint> keyPoints;
    //cout << "Extracting SIFT features of training images: Begin." << endl;
    for(int i=0; i<N_OF_TRAINING_IMAGES; i++){
        siftExtractor.operator()(trainImages[i], noArray(), keyPoints,
descriptorsOfTrainImages[i]);
    }
    //cout << "Extracting SIFT features of training images: End." << endl;



    //Merge all SIFT features from all training images to apply PCA
    Mat mergedTrainDescriptors;
    vconcat(descriptorsOfTrainImages, N_OF_TRAINING_IMAGES, mergedTrainDescriptors);


    //Apply PCA to merged SIFT descriptors of all training images
    pcaSIFT = PCA(mergedTrainDescriptors, Mat(), 0, pcaDimension);
    Mat projectedMergedTrainDescriptors;
    pcaSIFT.project(mergedTrainDescriptors, projectedMergedTrainDescriptors);


    //k-means clustering merged SIFT features of all training images
    Mat labels;
    int attempts = 5;
    Mat clusterMeans;
    //cout << "k-means clustering: Start" << endl;
    kmeans(projectedMergedTrainDescriptors, numberOfClusters, labels,
TermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 10000, 0.0001), attempts,
KMEANS_PP_CENTERS,
           clusterMeans);
    //cout << "k-means clustering: End" << endl;

    return clusterMeans;
}

/**
    Returns the histogram feature of an image

    @param inputIm input image
    @param codeWords codewords from the training data (cluster centers)
    @return histogram of codewords of the input image
*/

Mat aux::getHistogram(Mat& inputIm, Mat& codewords, SIFT& siftExtractor, PCA& pcaSIFT){

    //Initialize Histogram
    int numberOfHistogramBins = codewords.rows;
    Mat histogram = Mat::zeros(1, numberOfHistogramBins, CV_32FC1);

    //Extract SIFT features of the input image
    vector<KeyPoint> keyPoints; //dummy keyPoints variable
```

```
    Mat inputImDescriptor;
    siftExtractor.operator()(inputIm, noArray(), keyPoints, inputImDescriptor); //extract
SIFT features
    //Apply PCA to SIFT features of input image (reduce the dimensionality)
    Mat projectedInputImDescriptor;
    pcaSIFT.project(inputImDescriptor, projectedInputImDescriptor);

    //Construct the histogram
    int codewordIndex = 0;
    for(int i=0; i<projectedInputImDescriptor.rows; i++){
        //Quantize the vector, (get the index)
        Mat featureVectorToBeQuantized = projectedInputImDescriptor.row(i);
        codewordIndex = aux::getClosestCodewordIndex(featureVectorToBeQuantized,
codewords);
        //Update the histogram
        histogram.at<float_t>(0,codewordIndex) = histogram.at<float_t>(0,codewordIndex) +
1;
    }

    //Normalize the histogram
    histogram = histogram / (sum(histogram)[0]);

    return histogram;
}


int aux::getClosestCodewordIndex(Mat& inputFeatureVector, Mat& codewords){

    vector<float> distanceArray;

    //Get all distance between inputFeatureVector and all the codewords
    int numberOfCodewords = codewords.rows;
    for(int i=0; i<numberOfCodewords; i++){
        distanceArray.push_back(norm(inputFeatureVector, codewords.row(i)));
    }
    //Find the minimum distance and return the index
    int minIndex = min_element(distanceArray.begin(), distanceArray.end()) -
distanceArray.begin();
    return minIndex;
}

double aux::getErrorRate(Mat trueLabels, Mat predictedLabels){

    assert(("True Labels and Predicted Labels must be the same size", (trueLabels.rows ==
predictedLabels.rows) &&
                                                                      (trueLabels.cols ==
1) &&

(predictedLabels.cols == 1)));
    int numberOfSamples = trueLabels.rows;
    double numberOfErrors = 0;

    for(int i=0; i<numberOfSamples; i++){
        if(trueLabels.at<float_t>(0,i) != predictedLabels.at<float_t>(0,i)){
            numberOfErrors = numberOfErrors + 1;
        }
    }

    return numberOfErrors/numberOfSamples;
}

Mat aux::getConfusionMatrix(Mat trueLabels, Mat predictedLabels, int numberOfClasses){

    Mat confusionMatrix = Mat::zeros(numberOfClasses, numberOfClasses, CV_8UC1);
    int nPerClass = predictedLabels.rows / numberOfClasses;

    Mat predictedLabelsCopy = predictedLabels.clone();
    predictedLabelsCopy.convertTo(predictedLabelsCopy, CV_8UC1);

    for(int i=0; i<predictedLabels.rows; i++){
        confusionMatrix.at<uchar>(i/nPerClass, predictedLabelsCopy.at<uchar>(i,0)) += 1;
```

```
    }


    return confusionMatrix;
}



String aux::getClassName(int index){
    /*
     label 0 --> car_side
     label 1 --> butterfly
     label 2 --> faces
     label 3 --> watch
     label 4 --> water_lilly
     */
    if(index == 0){return "car_side";}
    if(index == 1){return "butterfly";}
    if(index == 2){return "faces";}
    if(index == 3){return "watch";}
    if(index == 4){return "water_lilly";}

    return "";
}
```

## auxFunctions.h

```
//CSCI574 HW5
//Ahmet Can Ozbek
//8401860152
#ifndef __CSCI574HW5__auxFunctions__
#define __CSCI574HW5__auxFunctions__

#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/nonfree/features2d.hpp>


using namespace cv;

namespace aux {
    //Functions
    //Reading Images and Constructing True Labels
    void readTrainImages(Mat* inputMatArray, int arraySize);
    void readTestImages(Mat* inputMatArray, int arraySize);
    void getTrueTrainingLabels(Mat& trueTrainingLabels, int size);
    void getTrueTestLabels(Mat& trueTestLabels, int size);
    String getClassName(int index);

    //Get the words
    Mat getWords(Mat* trainImages, Mat* descriptorsOfTrainImages, PCA& pcaSIFT, SIFT&
siftExtractor, int pcaDimension, int numberOfClusters);
    Mat getHistogram(Mat& inputIm, Mat& codewords, SIFT& siftExtractor, PCA& pcaSIFT);
    int getClosestCodewordIndex(Mat& inputFeatureVector, Mat& codewords);

    //Get Error Rate
    double getErrorRate(Mat trueLabels, Mat predictedLabels);
    //Get Confusion Matrix
    Mat getConfusionMatrix(Mat trueLabels, Mat predictedLabels, int numberOfClasses);
}


#endif /* defined(__CSCI574HW5__auxFunctions__) */
```