# CSE 152 : Introduction to Computer Vision, Spring 2018 – Assignment 5

### Instructor: Ben Ochoa

### Assignment Published On: Wednesday, May 23, 2018

### Due On: Saturday, June 9, 2018, 11:59 PM

## Instructions

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment contains only programming problems.
- All solutions must be written in this notebook
- Programming aspects of this assignment must be completed using Python in this notebook.
- If you want to modify the skeleton code, you can do so. This has been provided just to provide you with a framework for the solution.
- You may use python packages for basic linear algebra (you can use numpy or scipy for basic operations), but you may not use packages that directly solve the problem.
- If you are unsure about using a specific package or function, then ask the instructor and teaching assistants for clarification.
- You must submit this notebook exported as a pdf. All answers and results must be present in the .pdf file. Points will not be given for answers and results only present in the .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- It is highly recommended that you begin working on this assignment early.

## Introduction

In this assignment, we will have a look at some simple techniques for object recognition, in particular, we will try to recognize faces. The face data that we will use is derived from the Yale Face Database. The database consists of 5760 images of 10 individuals, each under 9 poses and 64 different lighting conditions. The availability of such standardized databases is important for scientific research as they are useful for benchmarking different algorithms.

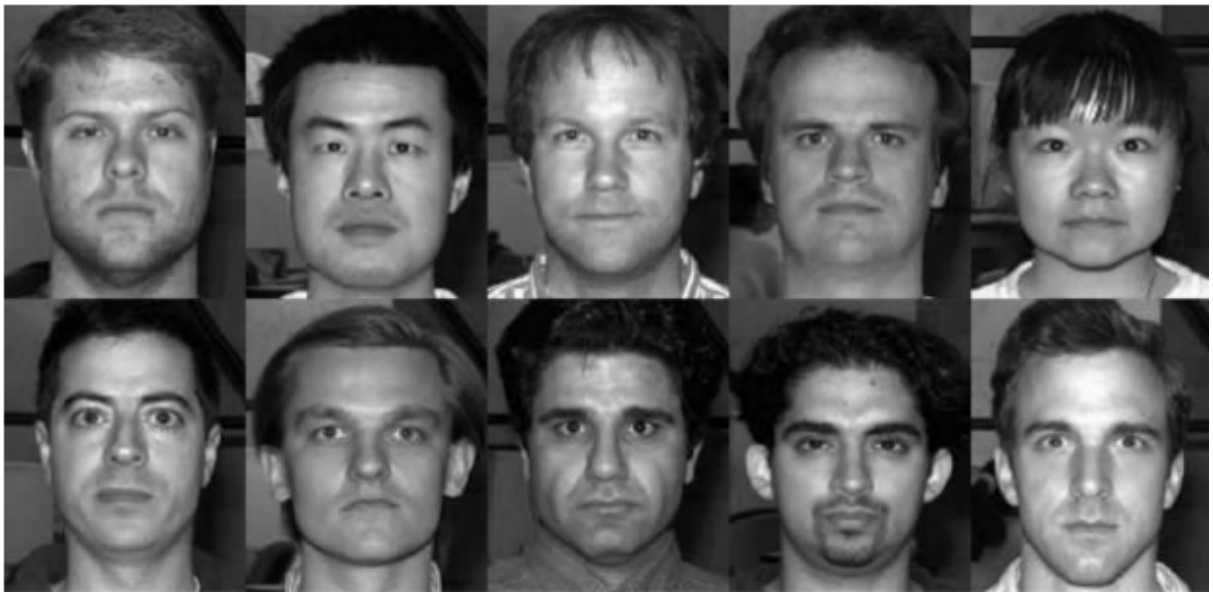

Figure1: The Yale face database B.

In this assignment, we will only use 640 images corresponding to a frontal orientation of the face. These faces are included in the file yaleBfaces.zip. You will find the faces divided into five different subsets. Subset 0 consists of images where the light source direction is almost frontal, so that almost all of the face is brightly illuminated. From subset 1 to 4, the light source is progressively moved toward the horizon, so that the effects of shadows increase and not all pixels are illuminated. The faces in subset 0 will be used as training images, and subsets 1 to 4 will be used as test images.

## Environment Setup

util.py requires cv2 library. Please run "conda install opencv" in Powershell if your are in Windows or Terminal if you are in Mac/Linux.

In [ ]:
```
import os, sys
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
from scipy.ndimage.filters import convolve
from scipy.ndimage.filters import gaussian_filter
import cv2
```

In [ ]:
```
def load_subset(subsets, base_path='data/yaleBfaces'):
    # INPUT:
    #   subsets:  the index of the subset to load, or a vector of sebset
    #      indices.  For example, loadSubset([2]) loads subset2, whereas
    #      loadSubset([0,1]) loads subset0 and subset1
    #   base_path:  the path to the yale dataset directory.  If left bla
    #     defaults to 'yaleBfaces'
    #
    # OUTPUT:
    #   imgs: a NXd matrix of images, where N is the number of images an
    #      the number of pixels in each image
    #   labels: a vector of length N, storing the person or class ID of
    #     image
    imgs = []
    labels = []

    for subset in subsets:
        directory = os.path.join(base_path, "subset" + str(subset))
        files = os.listdir(directory)
        for img in files:
            face = cv2.imread(os.path.join(directory,img), cv2.IMREAD_GR
            imgs.append(face)
            labels.append(int(img.split('person')[1].split('_')[0]))
    return imgs, labels
```

In [ ]:
```
def draw_faces(img_list, col=10):
    # Draw faces
    fig = plt.figure(figsize = (30,30))
    if len(img_list) < col:
        col = len(img_list)
        row = 1
    else:
        row = int(len(img_list)/col)
    for sub_img in range(1,row*col+1):
        ax = fig.add_subplot(row, col, sub_img)
        ax.imshow(img_list[sub_img-1], cmap='gray')
        ax.axis('off')
    plt.show()
```

# Problem 1 (Programing): Naive Recognition (15 points)

For this first problem, you will do face recognition by comparing the raw pixel values of the images. Let subset 0 be the train set, and report classification accuracy on test sets 1 to 4. Use the nearest neighbor classifier (1-NN) using the l2 norm (i.e. Euclidean distance).

1. (5 points) Once you have classified all images in a test set, report the average accuracy for each test set (percentage of correctly labeled images), i.e. 4 numbers.
2. (5 points) Comment on the performance, does it make sense? Is there any difference between the sets?
3. (5 points) Show any two misclassified examples and explain why these might have been misclassified.

In [ ]:
```
def predict_naive(test_imgs, train_img, label0):
    # test_imgs: list of Nxd array for Subset 1-4 as test cases
    # train_img: Nxd array for Subset0 as trainning set
    # label0: training set label
    # Your Code Here
    return label1, label2, label3, label4
```

In [ ]:
```
train_img, label0 = load_subset([0])
# Your Implememnation: You have to load the test set (Subset 1-4) by you
label1, label2 ,label3, label4 = predict_naive(test_imgs, train_img, lab
def evaluate(label1, label2, label3, label4):
    _, l = np.array(load_subset([1]))
    print("Subset 1 accuracy: " + str(np.count_nonzero((np.array(label1)
    _, l = np.array(load_subset([2]))
    print("Subset 2 accuracy: " + str(np.count_nonzero((np.array(label2)
    _, l = np.array(load_subset([3]))
    print("Subset 3 accuracy: " + str(np.count_nonzero((np.array(label3)
    _, l = np.array(load_subset([4]))
    print("Subset 4 accuracy: " + str(np.count_nonzero((np.array(label4)
evaluate(label1, label2 ,label3, label4)
```

In [ ]:
```
def show_mislabeled_img(imgs, labels):
    #Your Implementation here

show_mislabeled_img(imgs, labels)
```

## Problem 2 (Programing): k-Nearest Neighbors Recognition (10 points)

Instead of using a single nearest neighbor, sometimes it is useful to consider the consensus (majority vote) of k-nearest neighbours. Repeat Part 1.1 using k-nearest neighbor classifier (k-NN).

1. (4 points) Test the performance for each test dataset using k ∈ 1, 3, 5. Note that you already have the results for k = 1 from Part 1.
2. (4 points) Test the performance for each test dataset using the l1 norm rather than the l2 norm with k ∈ 1, 3, 5.
3. (2 points) Compare the performance of the classification with increasing k. Also comment on whether changing the distance metric influenced the results (if they did). Briefly justify the results observed.

In [ ]:
```
def predict_knn_l2(test_imgs, train_img, label0, k):
    # Your Implemenation Here

    return label1, label2, label3, label4
```

In [ ]:
```
def predict_knn_l1(test_imgs, train_img, label0, k):
    # Your Implemenation Here

    return label1, label2, label3, label4
```

In [ ]:
```
k = 3
train_img, label0 = load_subset([0])
# Your Implememnation: You have to load the test set (Subset 1-4) by you
def evaluate(label1, label2, label3, label4):
    _, l = np.array(load_subset([1]))
    print("Subset 1 accuracy: " + str(np.count_nonzero((np.array(label1)
    _, l = np.array(load_subset([2]))
    print("Subset 2 accuracy: " + str(np.count_nonzero((np.array(label2)
    _, l = np.array(load_subset([3]))
    print("Subset 3 accuracy: " + str(np.count_nonzero((np.array(label3)
    _, l = np.array(load_subset([4]))
    print("Subset 4 accuracy: " + str(np.count_nonzero((np.array(label4)
print("k = " + str(k))
print("By using l2 norm: ")
label1, label2 ,label3, label4 = predict_knn_l2(test_imgs, train_img, lab
evaluate(label1, label2 ,label3, label4)
print("By using l1 norm: ")
label1, label2 ,label3, label4 = predict_knn_l1(test_imgs, train_img, lab
evaluate(label1, label2 ,label3, label4)
```

# Problem 3 (Programing): Recognition Using Eigenfaces (25 points)

- (5 points) Write a function eigenTrain(trainset,k) that takes as input a N × d matrix trainset of vectorized images from subset 0, where N = 70 is the number of training images and d = 2500 is the number of pixels in each training image. Perform PCA on the data and compute the top k = 20 eigenvectors. Return the k × d matrix of eigenvectors W, and a d dimensional vector mu encoding the mean of the training images.
- (2 points) Rearrange each of the top 20 eigenvectors you obtained in the previous step into a 2D image of size 50 × 50. Display these images by appending them together into a 500 × 100 image (a 10 × 2 grid of images).
- (2 points) Explain the objective of performing PCA on the training images. What does this achieve?
- (2 points) Select one image per person from subset 0 (e.g., the 5 images person01 01.png, person02 01.png, ... , person10 01.png). Show what each of these images would look like when using only the top k eigenvectors to reconstruct them, for k = 1, 2, 3, 4, 5, ...10. This reconstruction procedure should project each image into a k dimensional space, project that k dimensional space back into a 2500 dimensional space, and finally resize that 2500 vector into a 50 × 50 image.

    - (10 points) Write a function called eigenTest(trainset,trainlabels,testset,W,mu,k) that takes as input :
    - The same N × d matrix trainset of vectorized images from subset 0
    - An N dimensional vector trainlabels that encodes the class label of each training image (e.g., 1 for person01, 2 for person02, etc.)

- An M × d matrix testset of M vectorized images from one of the test subsets (1-4)
- The output of PCA i.e. W and mu, and the number of eigenvectors to use k

Project each image from trainset and testset onto the space spanned by the first k eigenvectors. For each test image, find the nearest neighbor (1-NN) in the training set using an L2 distance in this lower dimensional space and predict the class label as the class of the nearest training image. Your function should return an M dimensional vector testlabels encoding the predicted class label for each test example. Evaluate eigenTest on each test subset 1-4 separately for values k = 1...20 (so it should be evaluated 4 × 20 times). Plot the error rate (fraction of incorrect predicted class labels) of each subset as a function of k in the same plot, and use the Python legend function add a *legend* to your plot.

- (2 points) Repeat the experiment from the previous step, but throw out the first 4 eigenvectors. That is, use k eigenvectors starting with the 5th eigenvector. Produce a plot similar to the one in the previous step. How do you explain the difference in recognition performance from the previous part?
- (2 points) Explain any trends you observe in the variation of error rates as you move from subsets 1 to 4 and as you increase the number of eigenvectors. Use images from each subset to reinforce your claims.

In [ ]:
```
imgs, labels = load_subset([0])
trainset = np.reshape(imgs,(70, 2500))
```

In [ ]:
```
def eigenTrain(trainset,k):
    # Your Implementation Here

    return eigvecs, mu
```

In [ ]:
```
def displayImages(k):
    # Display the 500 x 100 Image
    # Your Implementation Here
```

In [ ]:
```
def eigenTest(trainset,trainlabels,testset,W,mu,k):
    # Your Implementation Here
    # trainset:
    #    N × d matrix trainset of vectorized images from subset 0
    # trainlabels:
    #    An N dimensional vector trainlabels that encodes the class
    #    label of each training image
    # testset:
    #    M × d matrix testset of M vectorized images from one of
    #    the test subsets (1-4)
    # W:
    #    Coefficients of PCA
    # mu:
    #    Estimated mean
    # k:
    #    number of eigenvectors
```

# Problem 4 (Programing): Recognition Using Fisherfaces

# (20 points)

- (10 points) Write a function called fisherTrain(trainset,trainlabels,c) that takes as input the same N × d matrix trainset of vectorized images from subset 0, the corresponding class labels trainlabels, and the number of classes c = 10. Your function should do the following :
  - Compute the mean $mu$ of the training data, and use PCA to compute the first N − c principal components. Let this be $W_{PCA}$ .
  - Use $W_{PCA}$ to project the training data into a space of dimension (N − c).
  - Compute the between-class scatter matrix $S_B$ and the within class scatter matrix $S_W$ on the (N − c) dimensional space from the previous space.
  - Compute $W_{FLD}$ , by solving for the generalized eigenvectors of the (c−1) largest generalized eigenvalues for the problem $S_B w_i = \lambda_i S_W w_i$. You can use inbuilt functions to solve for the generalized eigenvalues of $S_B$ and $S_W$.
- The fisher bases will be a $W = W_{FLD} W_{PCA}$, where W is (c − 1) × d dimensional, $W_{FLD}$ is (c − 1) × (N − c) dimensional, and $W_{PCA}$ is (N − c) × d dimensional.
- (5 points) As in the Eigenfaces exercise, rearrange the top 9 Fisher bases you obtained in the previous part into images of size 50 × 50 and stack them into one big 450 × 50 image.
- (5 points) As in the eigenfaces exercise, perform recognition on the testset with Fisherfaces. As before, use a nearest neighbor classifier (1-NN), and evaluate results separately for each test subset 1-4 for values k = 1...9. Plot the error rate of each subset as a function of k in the same plot, and use the legend function in Python to add a *legend* to your plot. Explain any trends you observe in the variation of error rates with different subsets and different values of k, and compare performance to the Eigenface method. paper-link

In [ ]:
```
def fisherTrain(trainset, trainlabel, c):
    #Your Implementation Here
```