Introduction to Computer Vision (ECSE 415)
Assignment 3: Face Detection and Classification

Due date: 11:59PM, November 8th, 2021

This assignment will give you the opportunity to practice using eigenvectors to perform face detection and identification. You will work with a set of images containing faces, use principal components analysis (PCA) to represent them, then use that information to automatically detect those same faces in unseen data. You will then compare your implementation's performance to that of the Viola-Jones face detector (use publicly-available code; cite appropriately).

Please submit your assignment solutions electronically via the myCourses assignment dropbox. The submission should include: a single jupyter notebook. More details on the format of the submission can be found below. Submissions that do not follow the format will be penalized **10%**. Attempt all parts of this assignment. The assignment will be graded out of total of **75 points**. Note that you can use any of the OpenCV and scikit-learn functions shown during tutorial sessions for this assignment, unless stated otherwise.

For this assignment, each student will work alone and is expected to write their own code. (Academic integrity guidelines can be found here).

Note: Assignments received up to 24 hours late will be penalized by 10%. Assignments received more than 24 hours late will not be graded.

Instructions for the Report

1. Submit one single jupyter notebook for the whole assignment. It should contain everything (code, output, and answers to reasoning questions) [1]
2. Write answers section-wise. The numbering of the answers should match the questions exactly.
3. Comment your code appropriately.

[1] If you are facing some memory and RAM issues, you are free to submit multiple notebooks as long as you clearly mention the reason for doing so.

4. Do not submit input/output images. Output images should be displayed in the jupyter notebook itself.

5. Make sure that the submitted code is running without error. Add a README file if required.

6. We are expecting you to make a path variable at the beginning of your codebase. This should point to your working local (or google drive) folder. Ex. If you are reading an image as following:

```
img = cv2.imread('/content/drive/My Drive/Assi_2/images/
  Circles.png')
```

   Then you should convert it into the following:

```
    path = '/content/drive/My Drive/Assi_2/images/'
    img=cv2.imread(path+'Circles.png')
```

   Your path variable should be defined at the top of your jupyter notebook. While grading, we are expecting that we just have to change the path variable once and it will allow us to run your solution smoothly. Specify, your path variable in the README file.

7. The answers to the reasoning questions should be brief but comprehensive. Unnecessarily lengthy answers will be penalized. Use *markdown cell* type in jupyter notebook to write the answers.

8. If external libraries were used in your code please specify its name and version in the README file.

9. Run the whole jupyter notebook one last time using *Cell-> Run All*. This will run all cells in the jupyter notebook and show all outputs. We expect that when we run your jupyter notebook, we should be able to reproduce your results.

1 Data (7 Points)

For this assignment, you will have access to face images from the publicly available Georgia Tech Face Database [1]. This dataset is provided along with the assignment. The complete database contains images from 50 different subjects. All people in the database are represented by 15 color JPEG images. The pictures show frontal and/or tilted faces with different facial expressions, lighting conditions, and scale. A CSV file is provided with the dataset which maps image names with a label (subject ID) [2]. Each image has a different resolution. Convert images to gray-scale and resize all images to 128x192 pixel resolution. **(2 points)**

[2]Check Appendix-A for python code snippet to read the CSV file.

You now need to prepare your dataset for a face recognition and detection system. Randomly separate the images into training and test sets as described below. A new random selection should be made by your program every time the system is retrained [3].

- **Train set:** Randomly select 80% of the total images given. This will be used as the training set. **(1 point)**
- **Test set:** All the remaining images from the given dataset which are not used in the training set will be used as test images. **(1 point)**

Display total 10 random images from the training set along with their name and label . Plot histogram of the frequency of each image class (in this case the subject ID / label) distribution for both the training set and the testing set. **(3 Points)**

2 Eigenface Representation (25 Points)

You are now ready to create an eigenface representation for your training dataset through PCA. Please note that you are **not** allowed to use the in-built PCA function in OpenCV/Scikit-Learn. You should implement the efficient Snapshot method for PCA (covered in class, Lecture 8, Slide 55) from scratch using numpy. **(15 points)**

Plot the fraction of total variance against the number of eigenvectors [4] **(1 points)**. Plot the normalized variance (eigenvalues) against the eigenvector index used for computation. **(1 points)**. Do you need all the vectors to represent the data? Discuss **(3 points)**. Display the first 5 eigenfaces **(5 points)**.

3 Classification (23 Points)

For every testing image, find the nearest neighbour (L2 distance), and check whether both images belong to the same person. To estimate the accuracy of this approach, determine what fraction of your test images has a neighbour that is actually of the same person. Compute the accuracy both in the original high dimensional pixel space and then in the eigenspace, and compare the accuracy values. Would you expect there to be a significant difference? **(10 points)**

You will now use three different classifiers in the eigenspace (i) a linear SVM classifier, (ii) a RBF SVM classifier, (iii) Random Forest Classifier. Note that you are free to chose any hyper-parameters for these three classifiers. Use the training dataset to fit the classifier and the testing dataset to test the classifier.

[3] For debugging purposes, you can fix your random seed. This should allow you get same split during different runs.

[4] Refer to Tutorial 5 for more details.

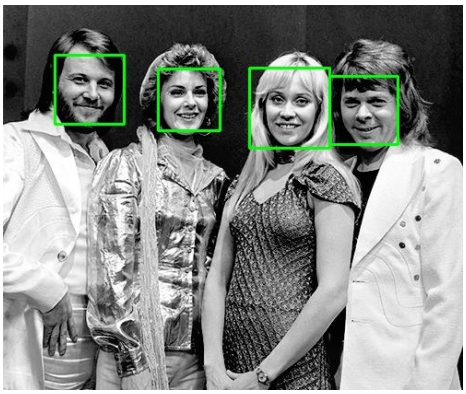

Figure 1: Face Detection: Example image with bounding boxes around the detected faces.

Compare the accuracy of these classifiers with the nearest neighbour classifier used previously **(10 Points)**.

Calculate per-class accuracy (Reference) for any one type of classifier and compare it against the calculated histogram of the frequency of each image class for the training dataset and the testing dataset in Section-1. Do you see any correlation between per-class accuracy and histograms? Write your observations. **(3 Points)**

4 Face Detection (20 Points)

You will now detect all the faces in the given group image (group_image.jpg) using PCA. Convert the image into grayscale. Use a sliding window to detect the faces. Set a threshold on the distance in eigenspace between the window contents and your training data (Refer to slide 63 of Lecture 8). Try different values of thresholds and use the one which gives you good results. Display your image with bounding boxes around the detected faces for the best threshold. (ex. Figure 1) [5] **(15 points)**.

Use an existing implementation of the Viola-Jones face detector, and compare the results with your detector. Comparisons should be made in terms of the number of true positives, false positives and false negatives (See Appendix-B). Display the image with bounding boxes around the detected faces. Under what conditions would you expect the Viola-Jones detector to work when PCA does not? **(5 points)**.

[5] Note that the emphasis is not on getting perfect detection but rather on your understanding and implementation. Do not worry if you do not get really good results. You can use any online available code for bounding box generation. Please cite the source for this in your report.

Appendix-A

```python
import pandas as pd

# path to csv file
# change this to your local path to mapping.csv
csv_path = "/home/raghav/Documents/ECSE_415/ECSE_415_F_2021/
    Assignment-3/mapping.csv"

# read csv file
df = pd.read_csv(csv_path)

# convert dataframe into two separate lists

# image - list -- contains name of images. Ex. ["S01_01", "S01_02",
     "S02_02", "S03_01", ...]
image = list(df["Image"])

# label - label -- contains subject ID / label. Ex. [1, 1, 2, 3,
    ...]
# Note how each image has an associated subject_ID / label.
# For Ex. "S01_01" has an associated label 1.
# This represents that S01_01 has class label 1.
# We will use these labels for classification purpose.
label = list(df["Label"])

# just for verification purpose
print(image)
print(label)
```

Listing 1: python code for reading a csv file

Appendix-B

A bounding box is considered as a true positive if it contains a face image otherwise, it is regarded as a false positive. A missed face (no detected bounding box around) is considered a false negative.