Homework 2
Due on October 10 at 11:59 pm

Python users can only use the libraries already imported in the HW2 Jupyter Notebook template

Problem 1: Simple Face Detector

You will implement a simple face detector that can detect faces in group photos of people

Data

- Training Dataset: You will use the LFWcrop (Labeled Faces in the Wild, cropped to the center portion of each image) database as a training dataset. You have been provided these images in the homework folder (`data/lfw_1000`)
- Testing Dataset: You have been provided four images of people in a group to use as your testing dataset (`data/groups`)

Methodology

1. Compute the first eigenface from training data. Read all the training images into a matrix, and then compute the first eigenvector for this matrix

You must now scan the group photos and identify all regions that match the pattern of the first eigenface the most. The test images are much bigger than 64x64, so every 64x64 region of the photo must be compared against the Eigenface.

2. Scan the image. Implement a way to consider every 64x64 region of the test image.
3. Detect faces in an image. M**atch** each 64x64 region against your calculated Eigenface. The **match score** between any region and the Eigenface is given by the **normalized dot product** between them. The dot product of the unrolled vector form of the Eigenface E and the unrolled image patch P is given by $\frac{E.P}{|P|}$. The locations of faces are likely where the **match score** has peaks when scanning the group photos.

The eigenface you found is fixed in size and can only be used to detect faces of the same size as the Eigenface itself. In the group photos, faces are sometimes different sizes. To solve this, scale the group images into many different sizes and scan them all with the eigenface.

4. Scale the image. Resize each group photo by a factor of 0.5, 0.75, 1.5, and 2.0. Therefore, there are a total of 5 versions of each group photo. Scan all versions, and calculate match scores for all of them, and locate peaks in all of them.
5. Merge nearby peaks. Sometimes, multiple peaks will occur in the same location, or within a few pixels of each other. In this case you can merge them.
6. Show the faces. Plot all the detected faces in the **original** group photo images.

Notes to consider:

- Some of the group photos are in color, whereas the Eigenface is in greyscale. You will have to convert the color photographs to greyscale by setting the grey values as the mean of the red, green, and blue values.
- For peaks that have been detected in the scaled group photos, you can find their corresponding positions in the original size image by de-scaling the coordinates. For example, if you found a peak at (X,Y) in the 2x image, the pixel positions in the original image are at (X/2,Y/2). The size of the face in the original image will be N/2 x M/2
- Additional heuristics can also be implemented for comparison of peak values from different scale factors. Appropriate thresholding for peak detection, additional scaling factors, etc. are for you to experiment with.

Problem 2: Boosting-based Face Detector

You will implement an Adaboost Classifier to classify between face images and non-face images. You may not use built-in MATLAB Adaboost classifier functions, or import additional Python libraries which create a classifier for you. You must implement it from scratch.

Data

- Training Dataset: You are given a training dataset of images (`data/boosting/train/face` and `data/boosting/train/non-face`). You will also be re-using the LFWcrop dataset (`data/lfw_1000`) from Problem 1 to generate multiple Eigenfaces.
- Testing Dataset: You are given a testing dataset of images (`data/boosting/test/face` and `data/boosting/test/non-face`).

Methodology

1. Compute the first K Eigenfaces. Compute the first K Eigenfaces from the LFWcrop dataset. You may need to resize these images to match the other boosting images. Set K=10 initially, but vary it appropriately to get the best results. This time, normalize the images to zero mean and unit variance before computing Eigenfaces.
2. Project the Face Images. Represent all the face images in `data/boosting/train/face` as a linear combination of the Eigenfaces calculated above. Therefore, write each face F_i as

$$F_i = w_{i,1}E_1 + w_{i,2}E_2 + \cdots + w_{i,K}E_K$$

   where E_j is the jth Eigenface and $w_{i,j}$ is the weight of the jth Eigenface composing the ith Face image. (Note that $w_{i,j}$ is simply the dot-product between F_i and E_j).

   Represent each face F_i in your training set by the weights for the Eigenfaces. Therefore,

$$F_i \rightarrow \{w_{i,1}, w_{i,2}, \ldots, w_{i,K}\}$$

3. Project the Non-Face Images. Similarly, represent all of the non-face images in your training data as linear combinations of the Eigenfaces, and use the weights of these Eigenfaces to represent these images. Therefore, for a non-face Image NF_i we have

   $$NF_i = v_{i,1}E_1 + V_{i,2}E_2 + \cdots + v_{i,K}E_K$$

   As before, the weights $v_{i,j}$ can be computed as dot-products, and the non-face images can be represented as

   $$NF_i \rightarrow \{v_{i,1}, v_{i,2}, \ldots, v_{i,K}\}$$

4. Classify between Face and Non-Face. The set of weights for the Eigenfaces for all images are the features representing these images. Using these weights as features, train an Adaboost classifier to classify between Face and Non-Face images.
5. Report your overall classification accuracy on the combined testing data in `data/boosting/test` by classifying these images using your Adaboost calssifier and comparing with the true labels. Vary K and report your observations.

Problem 3: Gender Detector

You will implement a gender detection system using the PCA dimensions from images.
You may not use built-in MATLAB PCA functions, or import additional Python libraries which extract PCA dimensions for you. You must implement it from scratch.

Data

- Training Dataset: You are given a training dataset of images split into male and female faces (`data/lfw_genders/male/train` and `data/lfw_genders/female/train`)
- Testing Dataset: You are given a testing dataset of images split into male and female faces (`data/lfw_genders/male/test` and `data/lfw_genders/female/test`)

Methodology

1. Load all data. Load the training data in `data/lfw_genders/male/train` and `data/lfw_genders/female/train`, change each image into a vector by reshaping it, and combine into an image matrix.
2. Find PCA dimensions. Find the K best PCA dimensions of this data by performing PCA analysis on the image matrix (calculate the eigenvectors and eigenvalues for the correlation matrix of the image matrix). Here $K \in \{50,100,200,300\}$

3. Select K. Plot all the eigenvalues of the correlation matrix for the image matrix, and suggest which value of K you think captures sufficient information from the training dataset.
4. Find the average Face. Calculate and plot the average male face and the average female face by finding the mean of all the male training images and all the female training images.
5. Detect the gender. Develop a simple gender detection algorithm based on these average faces as follows.
   i. Project your average male and female face on your K PCA dimensions.
   ii. Project all your testing images on your K PCA dimensions.
   iii. Classify each testing image as male or female by finding the distance between the weights of each testing image and the weights of your average male face and average female face.
   iv. Calculate the accuracy of classification across your testing data.
6. Test different K values. Calculate the accuracy of classification across your testing data for different values of $K \in \{50,100,200,300\}$ used for projection. Briefly explain the time vs accuracy trade-off for different values of K that you see.

Note: You *can* compare the projected faces of each testing face to the projected average male and Female faces as well. However, comparing weights is considerably faster, and these are equivalent if your eigenvectors from PCA are orthonormal. MATLAB's and NumPy's eig() function returns normalized eigenvectors, but svd() does not! You will have to manually normalize the eigenvectors if you use the svd() function.

7. Detect the gender. Develop a gender detection algorithm based on *all* training images as follows.
   i. Project *all* the training images into the K PCA dimensions, and *all* the testing images into the K PCA dimensions.
   ii. Calculate the average Euclidean distance between the weights of each test image and weights of *all* training male images and *all* training female images. Classify the test image based on which average Euclidian distance is smaller.
   iii. Calculate the accuracy of classification across your testing data.
8. Test different K values. Calculate the accuracy of classification across your testing data for different values of $K \in \{50,100,200,300\}$ used for projection. Argue about the time vs accuracy trade-off, which will be a lot more pronounced in this case.

Note: You *will* have to optimize your Euclidean distance calculation, or else your code will take too long to run. Matrix operations are your friend. Also, same as above, you *can* compare projected images directly, but comparing weights is much faster, as long as eigenvectors are normalized.

How To Submit

You must submit everything through canvas. If you use MATLAB, use the provided templates for your main code. If you use Python, use the provided Jupyter Notebook template for all problems, which is also available at following link-

https://colab.research.google.com/drive/1rnpNtAoWv2dqolDnPYKtpFqvkr_NK1R5?usp=sharing

Instructions for MATLAB:

You will submit a zip file containing all the required files to run your code and reproduce your results. Title your zip file as "HW2_FirstnameLastname_yourJHID.zip", where 'yourJHID' is your JHED ID.

For Problem 1: From the problem_1 folder, place your main implementation in the provided run_problem_1.m file. You can create as many auxiliary scripts as you want as long as we can get all of the plots we require by running only run_problem_1.m

Deliverables: `run_problem_1.m`

For Problem 2: From the problem_2 folder, place your main implementation in the provided run_problem_2.m file. You can create as many auxiliary scripts as you want as long as we can get all of the plots we require by running only run_problem_2.m

Place the requested observations and overall accuracy metrics in a pdf document titled `problem_2_answer.pdf`.

Deliverables: `run_problem_2.m, problem_2_answer.pdf`

For Problem 3: From the problem_3 folder, place your main implementation in the provided run_problem_3.m file. You can create as many auxiliary scripts as you want as long as we can get all of the plots we require by running only run_problem_3.m

Place the requested observations and overall accuracy metrics in a pdf document titled `problem_2_answer.pdf`.

Deliverables: `run_problem_3.m, problem_3_answer.pdf`

Instructions for Python:

You will have to use Jupyter Notebook for Python. We provided the template `.ipynb` file with the homework documents, and also provided a link to Google Colab. You have to submit the specific deliverables mentioned below in a zip file along with all the required files to run your code and reproduce your results. Title your zip file as "HW2_FirstnameLastname_yourJHID.zip", where 'yourJHID' is your JHED ID.

For Problem 1: Write your answers for problem 1 in the code cells under "Problem 1: Linear Algebra".

Deliverables: Code cells below "Problem 1 : Simple Face Detector" in `HW2_MLSP.ipynb`

For Problem 2: Write your answers for problem 2 in the code cells under "Problem 2: Boosting-based Face Detector".

Place the requested observations and overall accuracy metrics in a pdf document titled `problem_2_answer.pdf`.

Deliverables: Code cells below "Problem 2 : Boosting-based Face Detector" in `HW2_MLSP.ipynb`, `problem_2_answer.pdf`

For Problem 3: Write your answers for problem 3 in the code cells under "Problem 3: Gender Detector".

Place the requested observations and overall accuracy metrics in a pdf document titled `problem_3_answer.pdf`.

Deliverables: Code cells below "Problem 2 : Boosting-based Face Detector" in `HW2_MLSP.ipynb`, `problem_3_answer.pdf`