ECE420 Final Project Proposal

Tip of the tongue

Rutvik Sayankar, Charlie Plonski, Aahan Thapliyal

{rutviks2, plonski2, aahant2} @ illinois.edu

I. Introduction

Our team has observed a recurring issue amongst our members where we tend to struggle with recalling the names of individuals we encounter. As a remedy, we endeavor to develop an Android application that addresses this predicament. Our project comprises three facets. We aim to design an app that can track a user's face in real time. Secondly, we aim to recognize and classify faces whilst tracking the user. Finally, we aimed to incorporate speech recognition capabilities, such that when the subject within the frame verbalizes their name, the application shall record and associate the name with their corresponding facial features. We intend to construct the Fisherfaces & Eigenfaces algorithm from scratch, whilst utilizing native Android libraries for the audio processing component of our project.

II. Background Research

After spending time on the Assigned Project Lab we realized that both our implementation of Eigenfaces (which uses PCA) and using library functions lead to similarly poor-performing facial classification in dynamic lighting conditions. Therefore, for the final project, we are planning to transition over to Fisherface (which uses LDA) algorithm [1] and we would like to show how this algorithm's performance compares to the Eigenface algorithm. The Fisherface algorithm is aimed to be insensitive to large variations in lighting directions and facial expressions. Taking this even further, LDA focuses on finding a feature subspace that maximizes the separability between the groups. While PCA is an unsupervised (meaning uses an unlabeled dataset) dimensionality reduction technique, it ignores the class label and focuses on capturing the direction of maximum variation in the data set.

III. Block Diagram of Proposed System

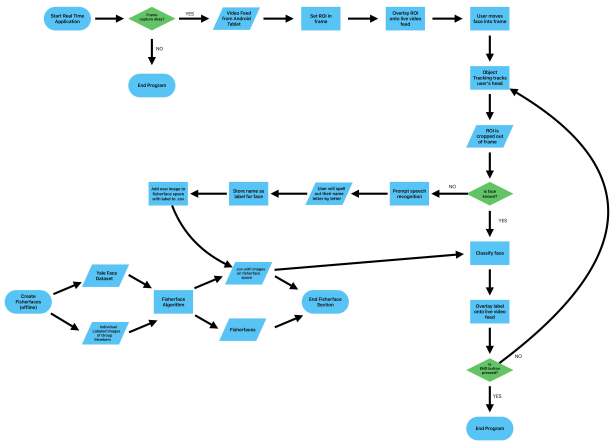

Fig. 1: Full Flow Chart of our project.

IV. Overview of the Fisherface algorithm

Fisherface algorithm uses the information of labeled training data set to build more reliable algorithm to perform dimensionality reduction on the images. The Fisherface algorithm is derived off the Fisher's Linear Discriminant (FLD) which selects W in such a way that the ratio of the between-class scatter and within-class scatter is maximized. Let the between-class scatter matrix be S_B and the within-class scatter matrix be S_W. Then both of them are defined as:

$$S_B = \sum_{i=1}^{c} N_i(\mu_i - \mu)(\mu_i - \mu)^T$$

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

where μ_i is the mean of the image of class X_i and N_i is the number of samples in class X_i. The Fisherface algorithm projects the image set into a lower dimensional space so that the resulting within-class scatter matrix S_W is non-singular. This is achieved by using Principle Component Analysis to reduce the dimension of the feature space to $N-c$, and then applying FLD to reduce the dimension to $c-1$. Resulting in:

$$W_{opt}^T = W_{fld}^T W_{pca}^T$$

where

$$W_{pca} = argmax_W |W^T S_T W|$$

$$W_{fld} = argmax_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W_T W_{pca}^T S_W W_{pca} W|}$$

W_{opt} is used to convert the images into the fisherface subspace which are then used to compare different images and classify them.

V. Overview of the Eigenface algorithm

- The eigenface recognition algorithm [2] is a technique for facial recognition that uses a mathematical approach to analyze and compare faces. The algorithm is based on Principal Component Analysis (PCA). The eigenface algorithm works by analyzing a large set of images of faces and identifying the most significant features, or "eigenfaces," that best represent the variations among the faces. These eigenfaces are essentially a set of mathematical vectors that represent the most common variations in facial features such as eyes, nose, mouth, etc. When a new face is presented to the system, the algorithm compares the facial features with the eigenfaces in the database and calculates a score for how closely the new face matches each eigenface. The system then combines these scores to generate a final match score that indicates how closely the new face matches the faces in the database.
- The approach to initializing face recognition involves acquiring a set of initial face images for training and calculating eigenfaces from these images. The highest eigenvalue images are kept to define the face space. As new faces are encountered, the eigenfaces can be updated or recalculated. The weight distribution of each known individual can be calculated by projecting their face images onto the face space. Then to recognize new face images, the system calculates weights by projecting the input image onto each of the M eigenfaces. The image is then checked to see if it is a face and classified as either a known person or unknown based on its weight pattern. Optionally, the eigenfaces and/or weight patterns can be updated, and if the same unknown face is seen multiple times, its weight pattern can be incorporated into the known faces.

Calculating Eignefaces:

- Given a training set of face images : $\sum_{m=1}^{M} \Gamma_m$, we find the average eigenface $\Psi = \frac{1}{M} \sum_{m=1}^{M} \Gamma_m$ and each image differs from the average by $\Phi_i = \Gamma_i - \Psi$

- The produced set is subjected to the principle component analysis which results in the set of M orthanormal vectors u_n. Where the kth vector is chosen such as $\lambda_k = \frac{1}{M}\sum_{n=1}^{M} u_k^T \Phi_n$ is maximum and subject to the restrains

$$u_l^T u_k = \delta_{lk} = \begin{cases} 1 & \text{if } l = k \\ 0 & \text{otherwise} \end{cases}$$

- The vector u_k and scalars λ_k are the eigenvectors and the eigenvalues of the covariance matrix $C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T$ where $A = [\Phi_1 \Phi_2 ... \Phi_M]$
- The matrix C, however, is N^2 by N^2, and determining the N^2 eigenvectors and eigenvalues is an intractable task for typical image sizes.
- Following this analysis, we construct the M by M matrix $L = A^T A$ where $L_{mn} = \Phi_m^T \Phi_n$ and find the M eigenvectors, v_l of L. These vectors determine linear combinations of the M training set face images to form the eigenfaces $u_l = \sum_{k=1}^{M} v_{lk} \Phi_k \qquad l = 1, ..., M$
- With this analysis the calculations are greatly reduced, from the order of the number of pixels in the images (N^2) to the order of the number of images in the training set (M)

Classifying Eigenfaces:

- For each known individual, calculate the class vector Ω_k by averaging the eigenface pattern vectors Ω calculated from the original images of the individual.
- Choose a threshold θ_ϵ, that defines the maximum allowable distance from any face class, and a threshold θ_ϵ, that defines the maximum allowable distance from face space.
- For each new face image to be identified, calculate its pattern vector Ω, the distances ϵ_i to each known class, and the distance ϵ to face space.
- If the minimum distance $\epsilon_k < \theta_\epsilon$, and the distance $\epsilon < \theta_\epsilon$, classify the input face as the individual associated with class vector Ω_k. If the minimum distance $\epsilon_k > \theta_\epsilon$, but distance $\epsilon < \theta_\epsilon$, then the image may be classified as "unknown," and optionally used to begin a new face class.
- If the new image is classified as a known individual, this image may be added to the original set of familiar face images, and the eigenfaces may be recalculated. This gives the opportunity to modify the face space as the system encounters more instances of known faces.

VI. Pseudocode

```
src = os.path.join(root_dir,'./YALE/centered/') #YALE Dataset
allFileNames = os.listdir(src)

...
# Reshape Data
...

# Perform PCA
eigenfaces = pca(data)

# Get 'U' from SVD Analysis
W = eigenfaces[0] # Eigenvectors
s = eigenfaces[1] # Eigenvalues
print(W.shape)
print(data.shape)
eigenfaces = data.T.dot(W)
eigenfaces=eigenfaces.T

# Normalizes the eigenfaces between 0 & 1
for i in range(eigenfaces.shape[0]):
    eigenfaces[i] =
eigenfaces[i]/np.linalg.norm(eigenfaces[i])
# Creating our Eigen projection matrix
magicmatrix = eigenfaces[skippedeigenfaces:numEigenfaces]
```

Fig. 2: Eigenface Pseudocode

```
# Extension for fisher faces

# Yale Data Set -> 3D class seperated matrix
# (Number of classes, Pics per class, Flattened size of images)
classData = np.zeros((15,picsperclass,newsize[0]*newsize[1]))
for i in range(165):
    classData[i//picsperclass][i%picsperclass] = origdata[i]

# Calculating optimization
Sbprojected = np.zeros((numEigenfaces- skippedeigenfaces,numEigenfaces-
Swprojected = np.zeros((numEigenfaces-skippedeigenfaces,numEigenfaces-skippedeigenfaces))

# Calculating SBprojected
mean  = mean.flatten()

# Calculating the between class covariance matrix
for i in range(numclasses):
    temp = classData[i]
    classmean = np.mean(temp, axis = 0)
    left  = magicmatrix @(classmean - mean.flatten()).reshape(-1,1)
    right = (classmean - mean).reshape(1,-1) @ magicmatrix.T
    Sbprojected += picsperclass * left @ right

# Calculating the within class covariance matrix
for i in range(numclasses):
    for j in range(picsperclass):
        classmean = np.mean(classData[i], axis = 0)
        left = magicmatrix @ (classData[i][j] - classmean).reshape(-1,1)
        right = (classData[i][j] - classmean).reshape(1,-1) @ magicmatrix.T
        Swprojected += left @ right

# Calculating generalized eigenvectors of between & within class covariance matrices
geneigvals, geneigvecs = eigh(Sbprojected, Swprojected, eigvals_only=False)
geneigvals = geneigvals[::-1]
geneigvecs =  geneigvecs[:,::-1]
geneigvecs = geneigvecs.T

# Finding the number of fisherfaces to use
for i in range(1,numEigenfaces):
    potentialFish = geneigvecs[:i]
    upperterm = potentialFish @ Sbprojected @ potentialFish.T
    lowterm = potentialFish @ Swprojected @ potentialFish.T
    upperdet = scipy.linalg.det(upperterm)
    lowdet = scipy.linalg.det(lowterm)
    temp[i] = upperdet

# Generating final fisherfaces
fishMat = geneigvecs[:np.argmax(temp)]
FinalMat = fishMat @ magicmatrix
```

Fig. 3: Fisherface Pseudocode

VII. Application In Action

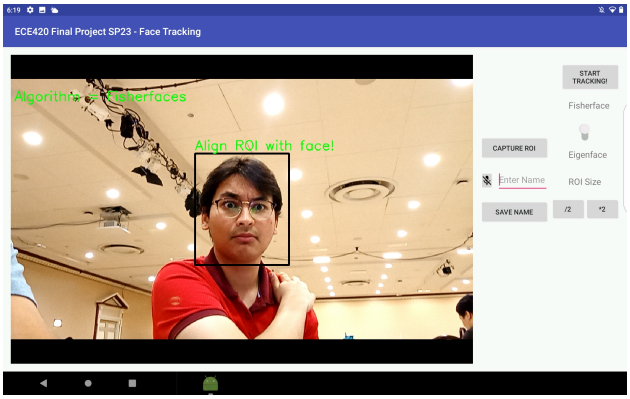

Fig. 4: Start Screen

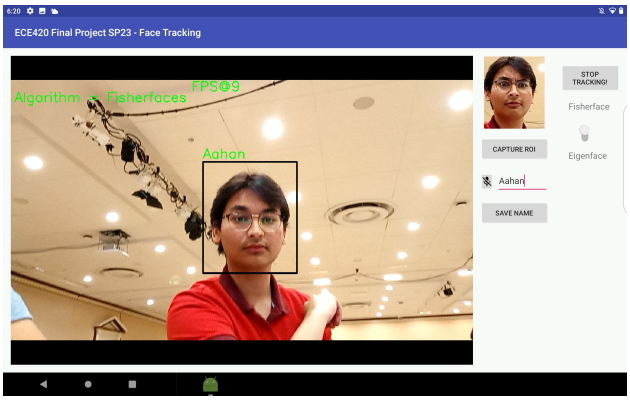

Fig. 5: Classification Screen

- The app starts off displaying a camera screen with an ROI box which can increase or decrease in size by pressing the increase/decrease button.
- It has a "Capture ROI" button which captures the ROI and projects it into eigenface space and fisherface space to be used for classification in real-time.
- The "Enter Name" box is used to enter the name associated with the image. The user can also enter the name by pressing the microphone next to it and saying the name into the microphone. The Save Name button then saves the name and associated projected images to the database that will be used later to compare to faces once the program starts.
- To start classifying a face the user can choose to use either the Eigenface or Fisherface algorithm by flipping the switch between them and then press "Start Tracking" once the face is within the ROI. The predicted name will be displayed on top of the ROI.

VIII. Validation Results

- **Testing with KMeans clustering** using YALE dataset (15 people 11 images per person). Reading from left to right top to bottom each 11 numbers correspond to different pictures of the same person and what cluster they were put in. (K=15 as there are 15 classes)

```
[ 2 14 14  9 14 14  5 14 14 14 14  0  0  0  9  0  0  5  0  0  0  0 11 11
 11  9 11 11  5 11 11  6 11  6  6  6 12  6  6  5  6  6  6  7  7  7 12
  7  7  5  7  7  7  7  2  4  4  9  4  4  4  4  4  4  4  3  3  3  9  3  3
  5  3  3  3  3 10  0  6  9 10  0  0  6  6  6 13 13 13 12 13 13  5 13
 13 13 13 10 10 10  9 10 10 10 10 10  6 10  8  8  8  8  8  8  8  8  8
  8 10  1  1 12  1  1 10  1  1  1  1 11 11 11 11 11 11  5 11 11 11 11  6
  6  6 12  6  6  5  6  6  6  6  0  0  0  9  0  0  5  0  0  0  0]
```

Fig. 6: Eigenfaces Kmeans clustering

```
[ 9  9  9  9  9  9  9  9  9  9  6  6  6  6  6  6  6  6  6  6  8  8
  8  8  8  8  8  8  8  8 14 14 14 14 14 14 14 14 14 14 14  0  0  0  0
  0  0  0  0  0  0  0 10 10 10 10 10 10 10 10 10 10 10  7  7  7  7  7  7
  7  7  7  7  7  1  1  1  1  1  1  1  1  1  1 11 11 11 11 11 11 11 11
 11 11 11  3  3  3  3  3  3  3  3  3  3  5  5  5  5  5  5  5  5  5  5
  5 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 13  2
  2  2  2  2  2  2  2  2  2  2  4  4  4  4  4  4  4  4  4  4]
```

Fig. 7: Fisheraces Kmeans Clustering

Clearly eigenfaces result in imperfect clusters as some of the 11 images of the same person get assigned to the clusters of other people. Whereas, fisherfaces gives perfect results as each of the 11 images of the same person for each of the 15 classes gets assigned to its own cluster

- **Testing by comparing distances** with YALE data set. After projecting to our eigen/fisher vectors we took one image and found the distance to all other images in our data set. The x-axis on these graphs is the number of the image we are calculating the distance from. The y axis is the distance of that image to the arbitrary image(in the range of 11-22) that we are comparing to.

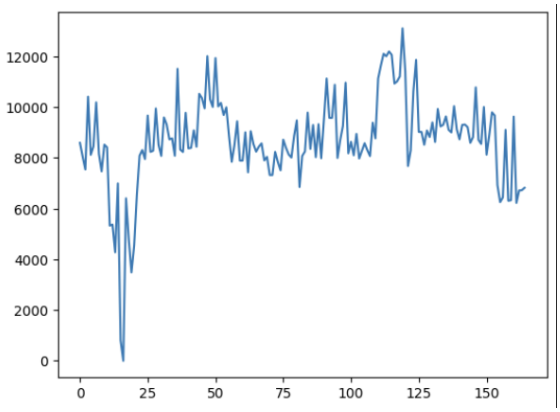

Fig. 8: Eigenfaces Distances

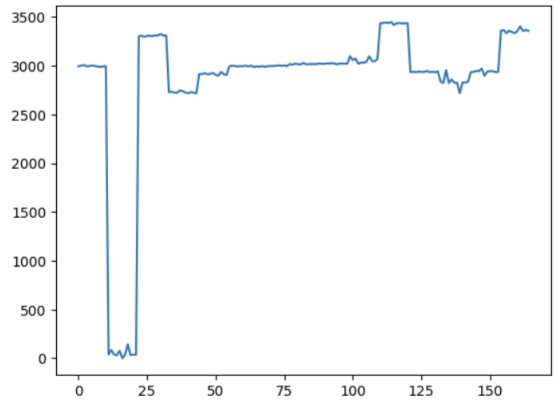

Fig. 9: Fisherfaces Distances

Clearly Eigenfaces can still tell the difference between pictures of the same people and pictures of different people, but there is not much margin for error. Whereas, Fisherfaces gives much closer distances than eigenfaces for images that are of the same person and much more distance for images that are of different people.

IX. Modification and Extension

- We would like to add multiple face detection and classification to the application so that it could work for real-life scenarios as well.
- Additionally, we would like to add automatic speech prompting to the app to automatically detect when a new name is spoken and store it in the prompt box allowing us to remove the microphone button on the app.

- Finally, improving the UI would make the app much more user-friendly, and not need any prior knowledge before using it as our app currently does.

X. Software/Hardware Documentation

- *MainActivity.java* was the main file that handled the following functions:
  - Processing the locally loaded .csv file that has the eigenfaces and fisherfaces.
  - Handles face tracking.
  - Handles real-time face recognition and classification.
  - Handles name and face storage.
  - Handles speech recognition.
- *GettingFaces.ipynb* handles generating the eigen and fisher faces from the YALE face classification training set. This script is run offline beforehand and generates the .csv which is downloaded locally on the Android tablet.
- Hardware Used:
  - Lenovo X606V Smart Tab M10 FHD Plus (4GB RAM, 128GB Storage)
  - NVIDIA SHIELD Tablet K1 (2 GB RAM, 128 GB Storage)
- Softwares Used:
  - Android Studio (Electric.Eel| 2022.1.1)
  - Jupyter Notebook
  - Visual Studio Code

References

[1] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 7, pp. 711–720, 1997.

[2] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of cognitive neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.