NANYANG TECHNOLOGICAL UNIVERSITY

EE4208 INTELLIGENT SYSTEM DESIGN

Assignment 1: Face Recognition with Principal Components Analysis (PCA)

Tutor: Dr Xu Yue Cong

Name	Matriculation Number	Design Group
1. Le Ngoc Canh	U1920690B	F41
2. Lin Yuxin	U1822583A	F43
3. Li Jiatong	U1822115J	F43

AY 21/22

Contents

Table of Figures

Contributions

Le Ngoc Canh	1. Collection of our dataset 2. Implementation of PCA Model (Collaboration with Lin Yuxin) 3. PCA performance with different number of components 4. Elbow diagrams between explained variance and number of components 5. Displaying name in real-time face recognition
	Report contribution: [Section 1, 2, 4.4], Slide Presentation
Lin Yuxin	1. Coding of dataset collection script and pre-processing function 2. Collection of our dataset 3. Implementation of PCA (Collaboration with Le Ngoc Canh) 4. Implementation of Face Detector (Haar / DNN) 5. Design and implementation of Haar-based Face Mask Detector 6. Integration of the whole code of the face recognition system 7. Design and coding of the solutions for handling mask and unseen faces
	Report contribution: [Section 3, 4, 5 (Collaboration with Le Ngoc Canh for 4.4), 6], Slide Presentation
Li Jiatong	1. Collection of our dataset 2. Merging of self-collected dataset and Olivetti dataset into single dataset 3. Display of dataset information and sample images as different requirements
	Report contribution [Section 7], Slide Presentation

1. Introduction

With the development of big data and computational capability, machine learning and artificial intelligent have become ever-increasing attractive field for research and development new solutions to real-life problems. Face recognition is an example of applications of machine learning and artificial intelligent. Particularly, it is one of the most outstanding developments of image analysis and computer vision fields. For example, in new generations of smart phones such as iPhone, users can unlock their phones using face recognition to avoid the frustrating part of inputting password manually. Additionally, automatic attendance check in schools or working premises is also an example of face recognition applications in real life. Moreover, they enhance police's capability to capture, recognize criminals and find missing people.

Face recognition deploys automated methodology to verify or recognize the identification of a person based on a biometric approach [1]. The methodology will classify the input image of the person as one of existing people inside the database. Due to the sensitivity of security, face recognition should possess two most important characteristics including high accuracy and fast processing model. The developer should consider carefully these two aspects to ensure that the model is balanced well between accuracy and processing time. The advantages of well-developed face recognition are reducing manpower for manual check-in, keeping a record for future checks, and ensuring the security in highly sensitive areas. However, there are some existing disadvantages as well such as large storage requirements, vulnerable and inaccurate detection, and potential personal privacy issues. For example, citizens in democratic countries concern their privacy and future use of their personal information.

2. Preparation and examples of dataset

For our dataset, we collected 30 people with 10 faces per people with 3 main orientations including front face, face rotated with 10-15 degrees to the left, and face rotated with 10-15 degrees to the rights. We display 30 distinct people with different names (label).

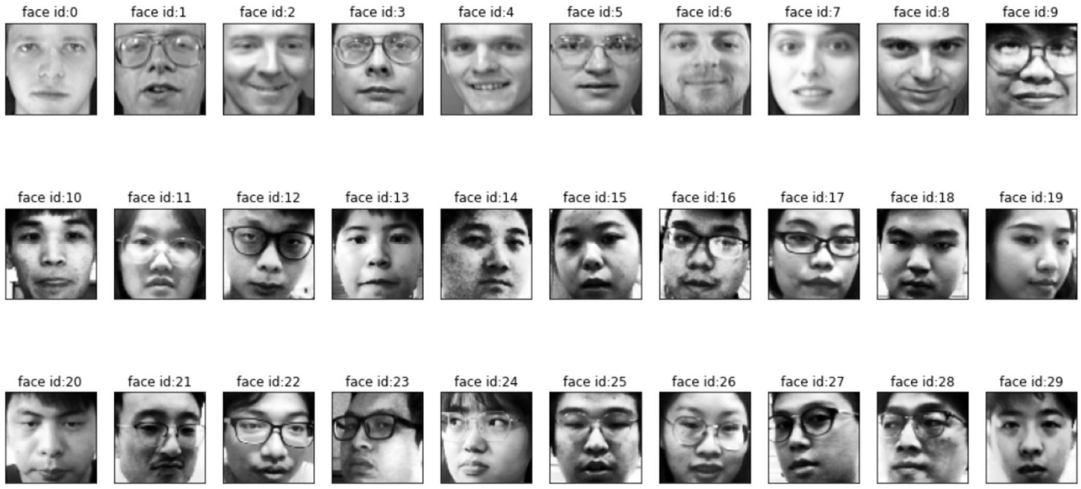

Figure 2.1 Representative from each class of 30 people

For each person inside dataset, we collected 10 faces per person. We displayed an example of 10 pictures from same person (same label).

Figure 2.2 Example of 10 faces per person from 4 classes

2.1. Pre-processing pipeline and data understanding

Some of our dataset is acquired under different environments. Although we have limited all of them to indoors, there can still exist covariates such as illumination variations. To ensure that all the images in the dataset are of the same style, they need to be first pre-processed.

The input is the raw face images from our dataset. The detailed steps are as follows:

1) Convert input image to grayscale.
2) Perform centre cropping to remove excess margins. (Compared with Olivetti dataset, the OpenCV DNN Face Detector yields a relatively large margin on the detected face images)
3) Resize the cropped image to 64 by 64
4) Perform Histogram Equalization (HE)
5) Normalize the image and divide by 255 to make it in the range of [0, 1]

```
def preprocessing(img, border=2):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # remove some border
    img = cv2.resize(img, (200, 200))
    height, width = img.shape[:2]
    img = img[border:height - border, border:width - border]
    img = cv2.resize(img, (64, 64))
    # Histogram Equalization
    img = cv2.equalizeHist(img)
    img = cv2.normalize(img, img, 0, 255, cv2.NORM_MINMAX)
    img = img / 255.
    return img
```

Figure 2.3 Pre-processing function including reading image, resizing, and normalization

To make the training set more diverse, we have merged our collected dataset with Olivetti dataset to increase the total number of subjects in the database.

Faces and their IDs are extracted from Olivetti dataset.

```
# fetch olivetti dataset for more faces
olive_data = fetch_olivetti_faces()['images'][0:olive_subject_total*10]
olive_target = fetch_olivetti_faces()['target'][:olive_subject_total*10]

# select from olive dataset according to config
selected_olivedata = []
selected_olivetarget = []
for k in range(0, olive_subject_total):
    used_imgae_startindex = 10 * k
    used_image_endindex = 10 * (k + 1)
    single_person_selected_olive_data = olive_data[used_imgae_startindex:used_image_endindex]
    single_person_selected_olive_target = olive_target[used_imgae_startindex:used_image_endindex]
    single_person_selected_olive_data = single_person_selected_olive_data[:olive_photo_per_subject]
    single_person_selected_olive_target = single_person_selected_olive_target[:olive_photo_per_subject]
    selected_olivedata.append(single_person_selected_olive_data)
    selected_olivetarget.append(single_person_selected_olive_target)

selected_olivedata = np.asarray(selected_olivedata).reshape(-1, 64, 64)
selected_olivetarget = np.asarray(selected_olivetarget).reshape(-1)
```

Figure 2.4 Getting 9 set of images from 9 people in Olievetti Dataset

Faces in our dataset are loaded and pre-processed. IDs for

```
# allocate id to subjects in our dataset
ntu_subject_id = olive_subject_total
# target list
ntudataset_target = []
ntudataset_label = []
# read data
for subject in ntudataset:
    subject_path = ntudataset_path + '/' + subject
    no_exp_face_img_filenames = os.listdir(subject_path)[:10]
    for imgfile in no_exp_face_img_filenames:
        img = cv2.imread(subject_path + '/' + imgfile)
        img = preprocessing(img, 20)
        noexpression_list.append(img)
        ntudataset_target.append(ntu_subject_id)
        ntudataset_label.append(subject)
    ntu_subject_id = ntu_subject_id + 1
data = np.concatenate((selected_olivedata, np.asarray(noexpression_list)), axis=0)
```

Figure 2.5 Read all data from dataset of group – 21 people with 10 images each person

```
target = np.concatenate((selected_olivetarget, np.asarray(ntudataset_target)), axis=0)
target_label = np.concatenate((selected_olivetarget, np.asarray(ntudataset_label)), axis=0)
print(target_label)
print("There are {} images in the dataset".format(len(data)))
print("There are {} unique targets in the dataset".format(len(np.unique(target))))
print("Size of each image is {}x{}".format(data.shape[1], data.shape[2]))
print("Pixel values were scaled to [0,1] interval. e.g:{}".format(data[0][0, :4]))
print("unique target number:", np.unique(target))
input("Press enter to start visualization and training...")

# visualize dataset
show_distinct_people(data, np.unique(target))

# You can playaround subject_ids to see other people faces
show_10_faces_of_n_subject(images=data, subject_ids=[0, 5, 10, 15])
```

Figure 2.6 Display information of dataset and visual display of some sample images in dataset

```
There are 300 images in the dataset
There are 30 unique targets in the dataset
Size of each image is 64x64
Pixel values were scaled to [0,1] interval. e.g:[0.30991736 0.36776859 0.41735536 0.44214877]
unique target number: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]
```

Figure 2.7 Dataset information display number of images, unique targets, size, and unique target number

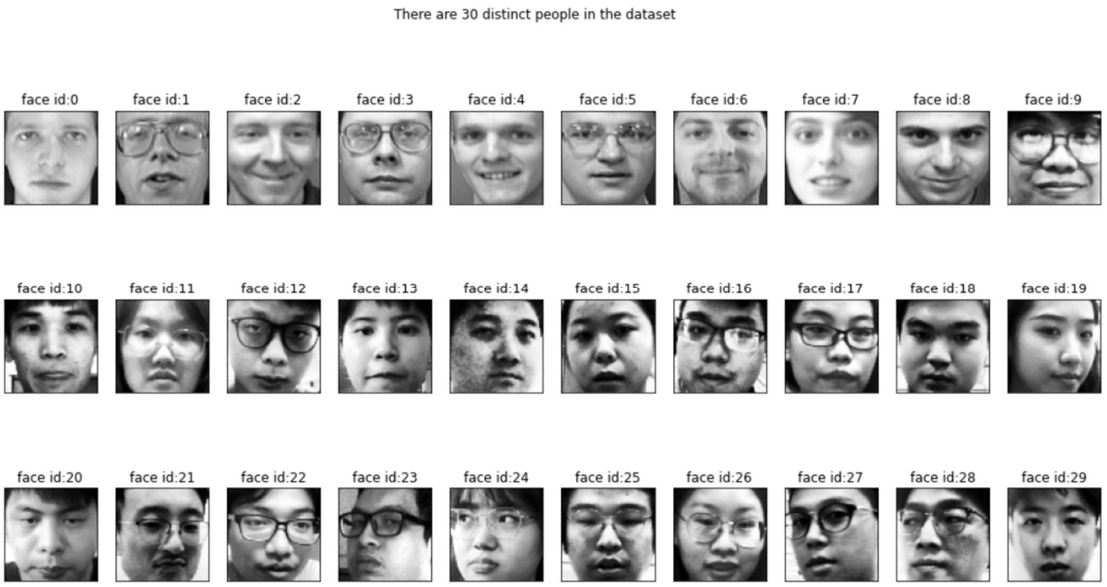

Figure 2.8 Representative image from each class

Figure 2.9 Example of ten images from each class of classes: 0,5,10, and 15

3. Face Recognition using Principal Component Analysis (PCA)

3.1. Overview of PCA

Principal Component Analysis (PCA) [2] is the process that computes the principal components and using them to perform a change of basis on the data.

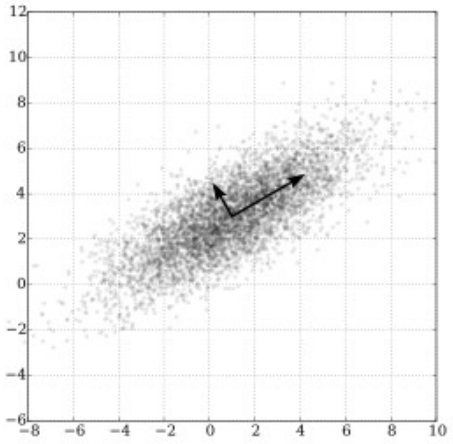

Figure 3.1 Illustration of PCA. The two principal components'' directions are indicated with arrows.

It can be shown that the principal components are the eigenvectors of the covariance matrix of the original data. Therefore, PCA is often computed using eigen decomposition. As the variance of the data along the eigen direction is decreasing from the 1st principal component, PCA is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variance as possible.

3.2. PCA Face Recognition and Eigenface

Based on the Principal Component Analysis (PCA), the main idea of Eigenface [3] is to decompose face images into a small set of characteristic feature images called eigenfaces.

The eigenfaces, which may also be referred to as the principal components of the original images, function as the orthogonal basis of a linear subspace called "eigenspace" or "face

space". Recognition is performed by projecting a new face image onto this subspace and comparing its position with those of the known ones.

For Eigenface, it can be divided into two stages: 1) training stage and 2) recognition stage.

3.2.1. Training Stage

In training stage, we are given a set of training 2D face images

$$\{F_1, F_2, \dots, F_n\}$$

of size $M \times M$.

The first step is to resize the n $M \times M$ faces into column vectors of length M*M.

After the resize, the faces become

$$\{f_1, f_2, \dots, f_n\}$$

With these n faces, we can obtain the average face:

$$f_{\text{ave}} = \frac{1}{n}\sum_{i=1}^{n} f_i$$

Then, we can compute the covariance matrix:

$$C = \frac{1}{n}\sum_{i=1}^{n} (f_i - f_{ave})\,(f_i - f_{ave})^T$$

After change of basis, the faces are like

$$f_i - f_{ave} = w_{i1}v_1 + w_{i2}v_2 + \cdots + w_{iK}v_K$$

Where $v_1, \dots, v_K$ are the eigenvectors (eigenfaces or principal components) of C.

Then, we can then perform dimension reduction. As of this step, only m eigenfaces with m largest eigenvalues of each face are kept.

$$f_i - f_{ave} \simeq w_{i1}v_1 + w_{i2}v_2 + \cdots + w_{im}v_m$$

Where $m \ll K$.

Finally, for each face $f_i - f_{ave}$, take dot product with each eigen vector:

$$(f_i - f_{ave})^T v_1 = w_{i1}$$

$$(f_i - f_{ave})^T v_2 = w_{i2}$$

$$\vdots$$

$$(f_i - f_{ave})^T v_m = w_{im}$$

Then for each face f_i, $w_i = [w_{i1}, w_{i2}, \dots, w_{im}]^T$. This will be used for final identification

3.2.2. Recognition Stage

Following the same procedure in training stage by calculating the difference between the input face and average face, we can obtain the input for classifier.

Classifiers such as SVM, Bayes Classifier, k-NNR can be used. In this project, we choose SVM as our classifier for identification.

3.3. SVM explanation

In this project, SVM is selected as the classifier for identification.

Support Vector Machine (SVM) is a linear model that is frequently adapted in the tasks of classification and regression. With non-linearities introduced by kernel functions, it can solve linear and non-linear problems and work well for many practical problems.

The idea of SVM is to create a line (or a hyperplane) which separates the data into different classes. However, for separable two-class classification problem, there exists two cases where the two classes are linearly separable and non-linearly separable.

If we can find a line (hyperplane) to separate the two classes, then they are said to be linearly separable.

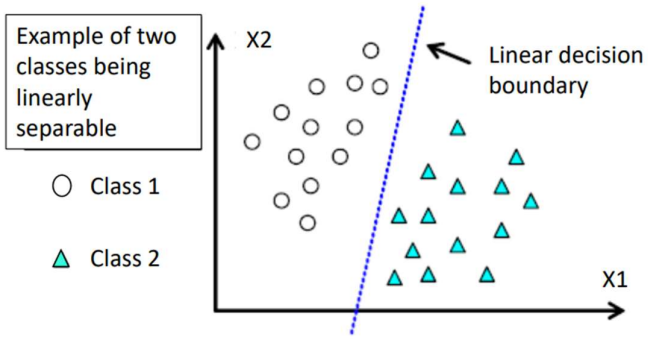

Figure 3.2 Example of two classes being linearly separable.

For some other cases, the two classes are separable but not linearly separable (e.g. in the example below). We say that the two classes are non-linearly separable.

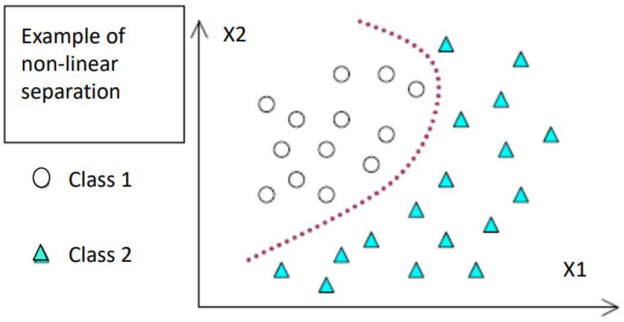

Figure 3.3 Example of two classes being non-linearly separable

For linearly separable case, there exists multiple lines (hyperplanes) that can separate the two classes.

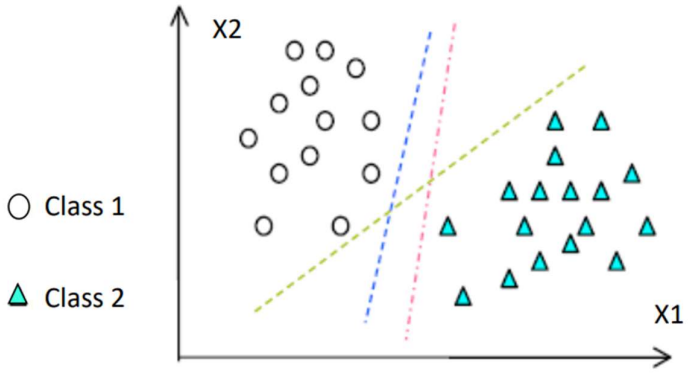

Figure 3.4 Multiple possible linear separations

However, what we want is not only a well-fitted model on the training set given, but a robust model that can also generalize well on unseen data. Among all the infinite linear separators,

there is one which has the best generalization capability - the SVM.

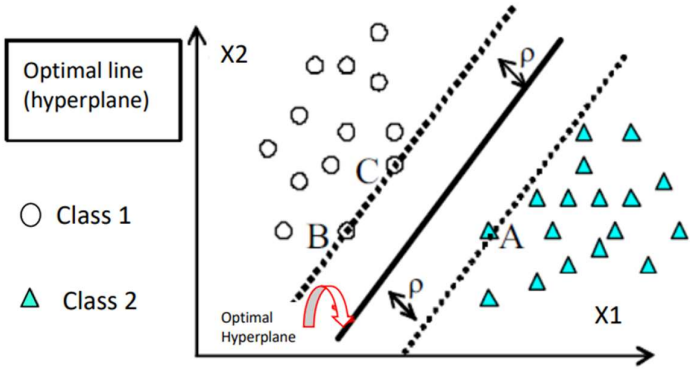

Figure 3.5 SVM illustration.

In figure 3.5 , A, B and C are the support vectors, and p is the maximum margin of separation from the optimal hyperplane.

According to Cover's Theorem on the Separability of Patterns (1965), a complex pattern classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space. Therefore, to handle the non-linearly separable case, kernel functions are introduced to bring more non-linearity by casting the original data from a lower dimension to a higher dimension.

Common kernel functions include polynomial, radial-basis function (RBF), and multi-layer perceptron. The mathematical forms are as follows:

$$\text{Polynomial: } G(x_1, x_2) = (1 + x_1' x_2)^p$$

$$\text{RBF: } G(x_1, x_2) = exp\ (\frac{-\|x_1 - x_2\|}{2\sigma^2})$$

$$\text{Multi-layer perceptron: } G(x_1, x_2) = tanh\ (p_1 x_1' x_2 + p_2)$$

4. Face Recognition Pipeline

To perform face recognition and identify given subjects, the first step is always to use a face detector to detect the faces as input. Once we have obtained the inputs, we can pass them to our trained model for recognition. In this COVID situation, the mask status is also of great concern to us. Therefore, to detect if the subjects are wearing masks properly, as well as recognize their identities, are two challenging tasks for us.

The whole face recognition pipeline is illustrated as follows:

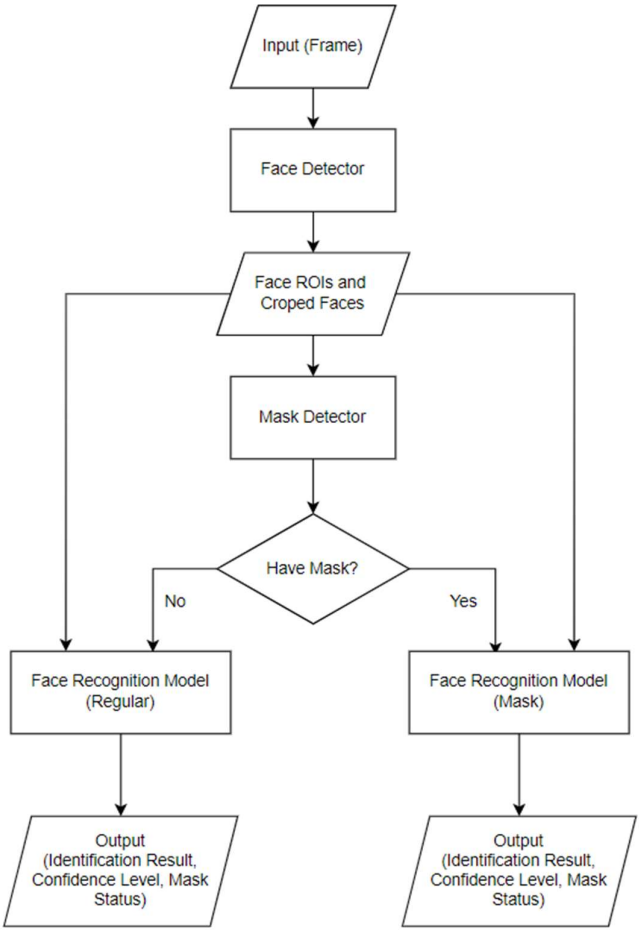

Figure 4.1 Face Recognition Pipeline

When a frame is given as input, it will first go through the Face Detector to detect the ROIs of faces. The cropped faces are passed to mask detectors to detect if the subject is wearing a mask. For the two different cases of mask status, two different face recognition models are used to generate the final output.

In summary, our face recognition pipeline contains 3 main steps:

1) Face Detection
2) Face Mask Detection
3) Identification

The details are covered in Section 4.1, 4.2, 4.3 and 4.4.

4.1. Face Detection

In order to perform face detection, a face detector is required. In OpenCV, there exists multiple face detectors with different pros and cons for application in different scenarios. In our experiment, we have mainly studied the Haar Cascade Detector and OpenCV DNN Detector (based on ResNet10).

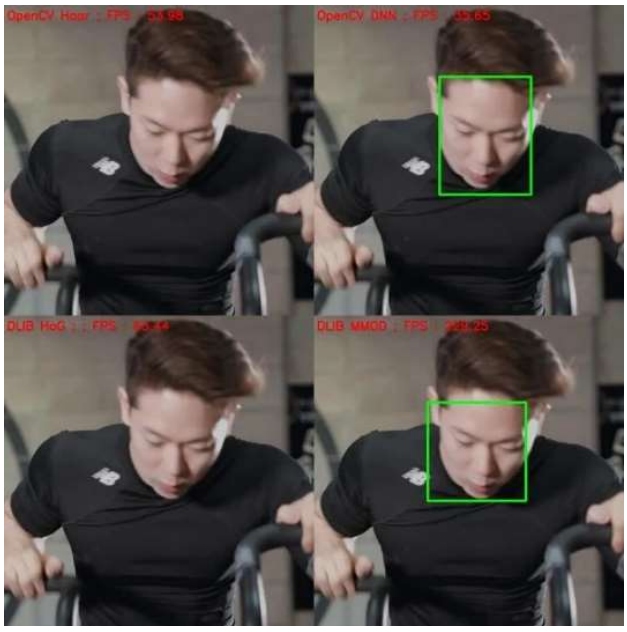

Figure 4.2 OpenCV Haar, DNN, HoG, MMOD face detectors. Both Haar and HoG failed to cope with pose variation in the image.

Haar Cascade based Face Detector was the state-of-the-art in Face Detection for many years since 2001, when it was introduced by Viola and Jones.

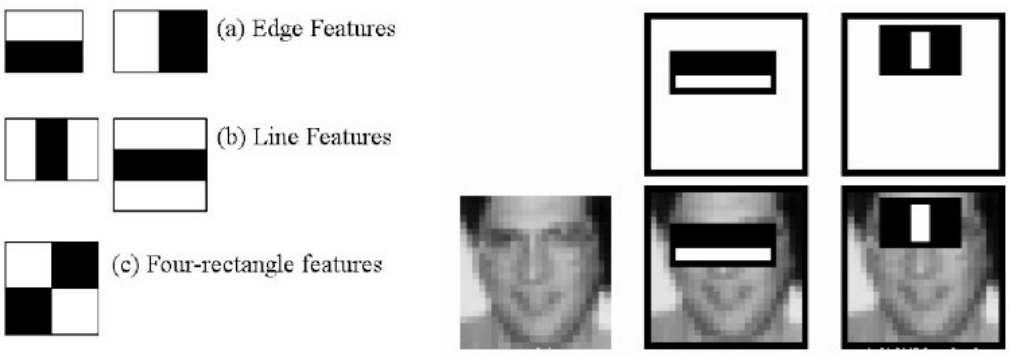

Figure 4.3 Haar Classifiers and illustration of Haar Cascade face detection

As a classical method, Haar Cascade Face Detector is not CPU/GPU intensive. With its simple architecture, it works real-time on mid- and even low-end CPUs and is able to detect faces in multiple scales.

However, in out testing, Haar Cascade Face Detector has a lot of false detections. Furthermore, the robustness of Haar Cascade Face Detector towards occlusion and pose variations is very limited.

As a result, we turned to DNN Face Detector implemented in OpenCV. This model was included in OpenCV from version 3.3. It is based on Single-Shot-Multibox detector based on ResNet-10 as backbone.

Convolutional stage	Output size	Layer
Conv1	200×150	7×7, 64, stride 2
		3×3 max pool, stride 2
Conv2_x	100×75	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}$
Conv3_x	50×38	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}$
Conv4_x	25×19	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}$
Conv5_x	13×10	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}$

Figure 4.4 Architecture of ResNet 10

Among the 4 face detectors above, the OpenCV DNN Face Detector has the best accuracy. Since the model is relatively small, it can also run at real-time on CPU. It has very impressive performance on pose variation, occlusion, etc.

4.2. Face Mask Detection

In this global pandemic of COVID, it is necessary to handle the case where face masks are present, partially covering the subjects' faces. Also, it is necessary to detect whether one is wearing face mask or not and take safety measures.

Although there already exists deep learning-based face mask detectors with state-of-the-art performance, the computational cost is heavy. As a result, such face mask detectors are not applicable for real time use, especially for devices with limited computing power, for example, embedded development boards without dedicated GPU such as Raspberry Pi. Therefore, we have designed a face mask detector using classical approach. The idea behind is quite intuitive and simple, and it works pretty well.

When a mask is not present, both the nose and eyes are present. When a mask is present, the nose of the subject is covered under the mask and the eyes are exposed.

Therefore, we have designed a Haar-based method which uses OpenCV Haar Cascade Detector to perform mask detection based on the status of presence of eyes and nose.

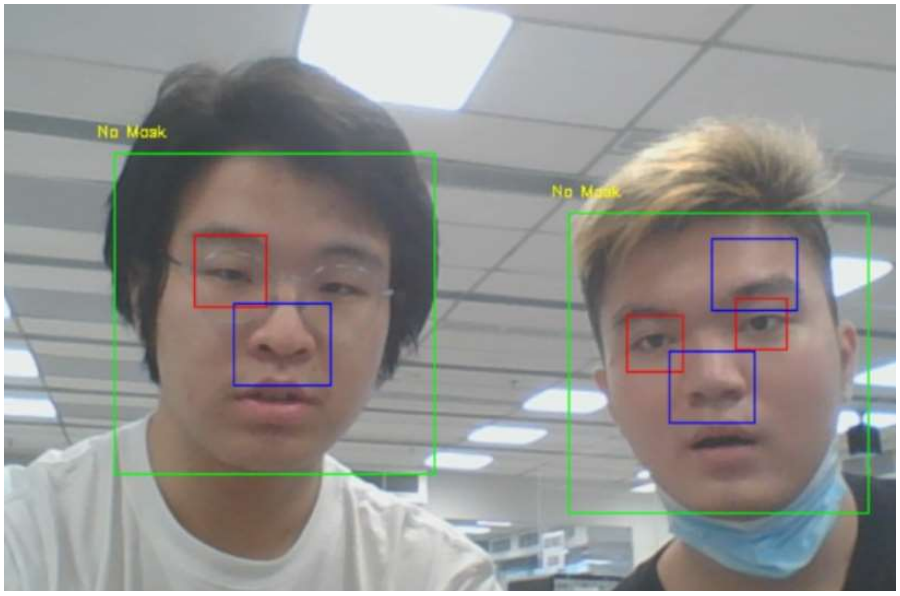

Figure 4.5 When 2 of us not wearing masks. Eyebrow is detected as "nose" as well

When not wearing masks, the noses are detected and highlighted in blue boxes, while the eyes are highlighted in red boxes.

However, since one shortcoming of Haar Detector is that it will generate false detections, we have implemented a relative position comparison of eyes and noses to avoid false noses.

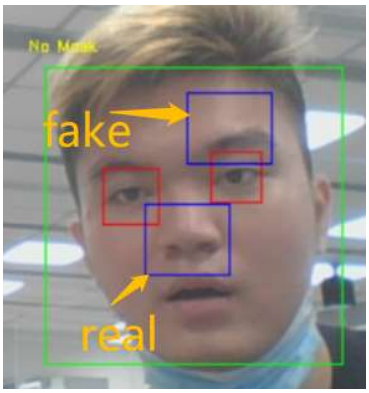

Figure 4.6 Fake "nose" and real nose

The logic is straightforward. In each face ROI, we compare the relative positions of detected eyes and noses. If there exists no detected nose that is lower than detected eyes, we assume that there is no real nose detected, but likely a false detection of other regions on the face.

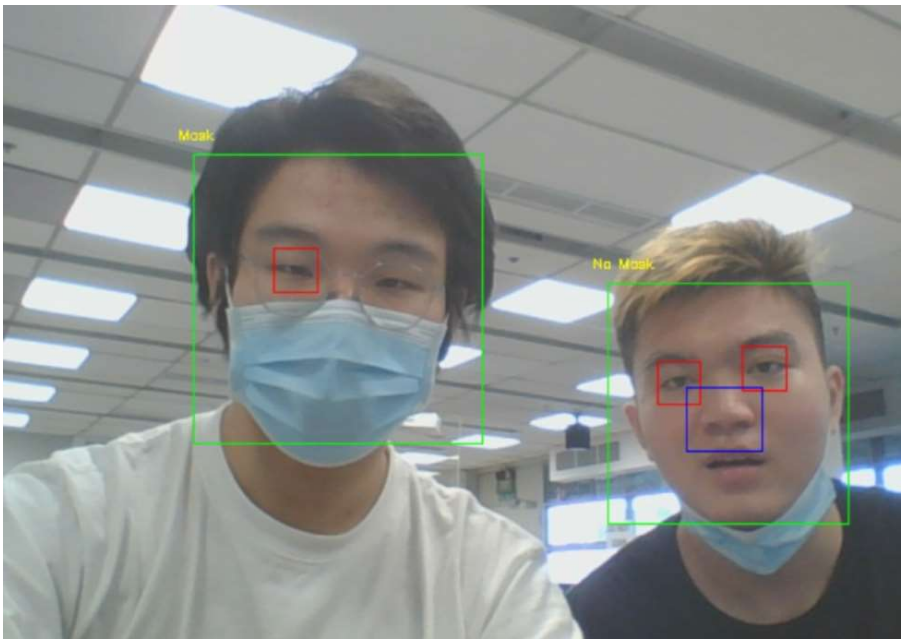

Figure 4.7 When mask is present and is not present

When a mask is present, nose will be covered so that the detection result will be empty. Since only eyes are detected, we can safely assume that the person is wearing a mask.

4.3. Identification

After obtaining both the cropped face images and mask status, the last step in face recognition, identification, can finally be done.

In our case, we have trained two separate models for handing 1) regular case without mask 2) subject wearing mask. Compared with the regular model, the model for face recognition with mask is trained using the upper half of the images.

Figure 4.8 Training images for 1) regular model without mask 2) model for subject wearing mask.

The cropping is done at runtime as part of the preprocessing. Therefore, we don't need to prepare a separate dataset beforehand.

When we are performing real-time recognition, the mask detector will return the mask status and decide which model is going to be used for identification. As we have branched 2 subroutines for handling both non-mask and mask cases using 2 separate models, our proposed face recognition system can well-handle the case when masks are present.

4.4. Our PCA Face Recognition Model: Eigenface + SVM

Our PCA Face Recognition is based on Eigenface and SVM. The detailed implementation are as follows:

4.4.1. Visualization and Parameter Tuning

To visualize the generated Eigenfaces as well as tuning the optimal $n\_components$ for PCA, we need to train PCA models with different $n\_components$ and compare their performance. Train and test set are randomly split from dataset with $test\_size = 30\%$.

```
# flatten as input for PCA model
X = data.reshape((data.shape[0], data.shape[1] * data.shape[2]))
print("X shape:", X.shape)

X_train, X_test, y_train, y_test = train_test_split(X, target, test_size=0.3, stratify=target, random_state=0)
print("X_train shape:", X_train.shape)
print("y_train shape:{}".format(y_train.shape))

y_frame = pd.DataFrame()
y_frame['subject ids'] = y_train
y_frame.groupby(['subject ids']).size().plot.bar(figsize=(15, 8), title="Number of Samples for Each Classes")
```

Figure 4.9 Data splitting of train and test dataset for processing

Figure below visualizes the number of training images for each class.

```
X shape: (300, 4096)
X_train shape: (210, 4096)
y_train shape:(210,)
```

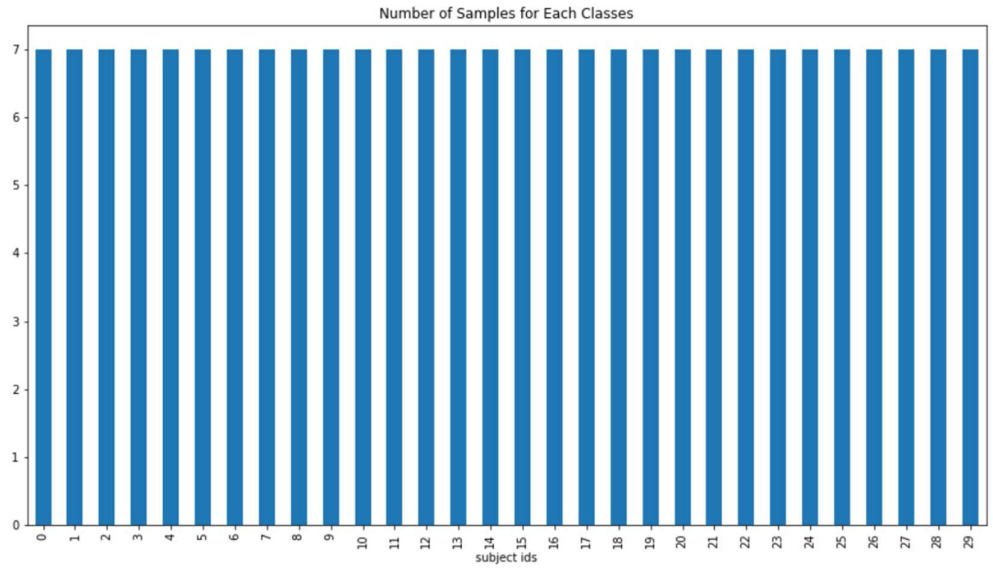

Figure 4.10 Train dataset information and display the number of images in each class of train dataset

To visualize our PCA results in 2D, PCA with 2 components is performed.

```
pca = PCA(n_components=2)
pca.fit(X)
X_pca = pca.transform(X)

# plot 10 ppl pca
number_of_people = 10
index_range = number_of_people * 3
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(1, 1, 1)
scatter = ax.scatter(X_pca[:index_range, 0],
                     X_pca[:index_range, 1],
                     c=target[:index_range],
                     s=10,
                     cmap=plt.get_cmap('jet', number_of_people)
                     )

ax.set_xlabel("First Principle Component")
ax.set_ylabel("Second Principle Component")
ax.set_title("PCA projection of {} people".format(number_of_people))
fig.colorbar(scatter)
plt.show()

# show avg face
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.imshow(pca.mean_.reshape((64, 64)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Face')
plt.show()
```

Figure 4.11 PCA Transformation, plotting 10 people PCA, and show average faces

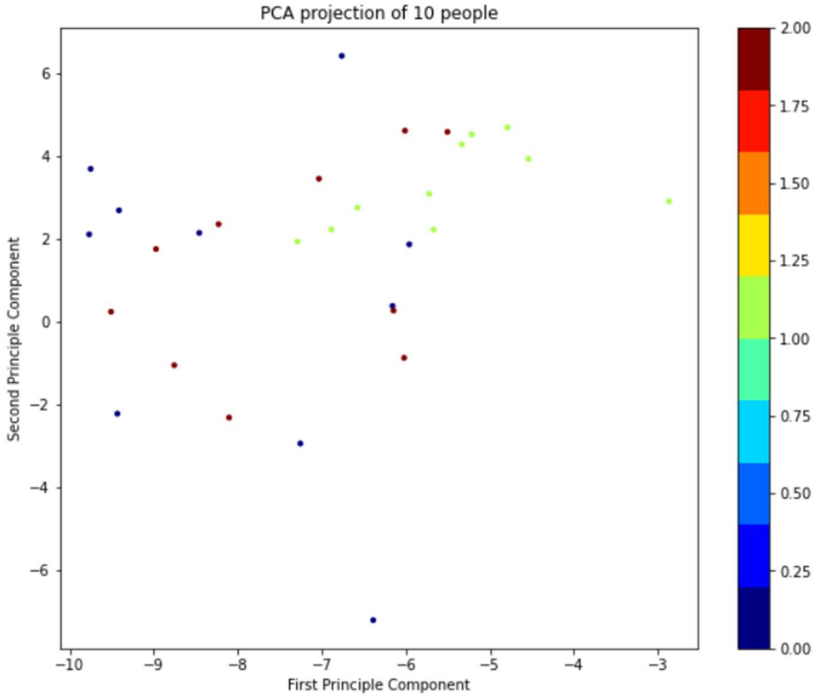

Figure 4.12 PCA projection of 10 people with 2 components

Figure 4.13 Average face with 2 components in PCA

Then, Eigenfaces of 30 people with 30 components in PCA are visualized.

```
n_components = 30
pca = PCA(n_components=n_components, whiten=True)
pca.fit(X_train)
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.imshow(pca.mean_.reshape((64, 64)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Face')
plt.show()

# show eigenface
number_of_eigenfaces = len(pca.components_)
eigen_faces = pca.components_.reshape((number_of_eigenfaces, data.shape[1], data.shape[2]))

print("num of eigenfaces:", number_of_eigenfaces)
cols = 10
rows = int(number_of_eigenfaces / cols)
fig, axarr = plt.subplots(nrows=rows, ncols=cols, figsize=(15, 15))
axarr = axarr.flatten()
for i in range(number_of_eigenfaces):
    axarr[i].imshow(eigen_faces[i], cmap="gray")
    axarr[i].set_xticks([])
    axarr[i].set_yticks([])
    axarr[i].set_title("eigen id:{}".format(i))
plt.suptitle("All Eigen Faces".format(10 * "=", 10 * "="))
plt.show()
```

Figure 4.14 Code for showing eigenface with 30 components in PCA

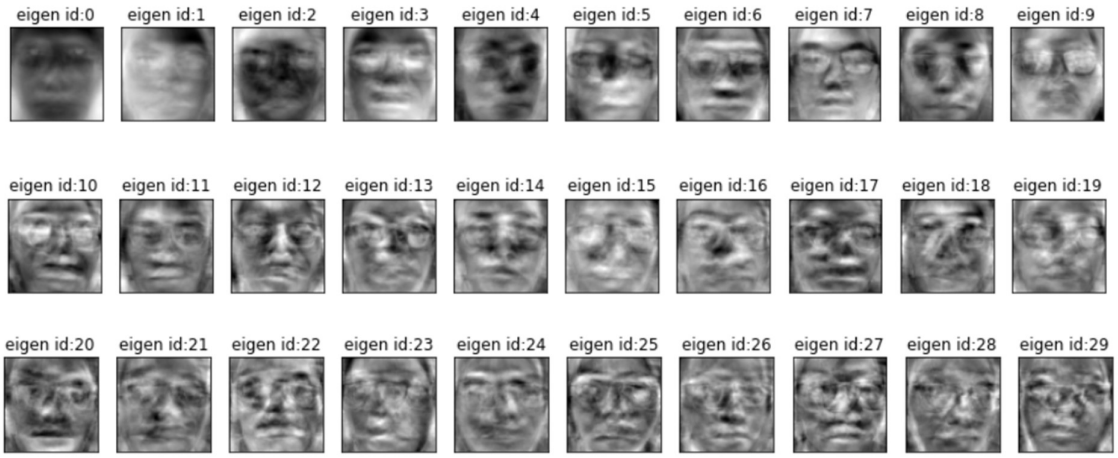

Figure 4.15 Display of eigenface with 30 components in PCA

To better understand the variance preservation with different $n\_compoments$, PCA number of components vs. explained variances is visualized.

From the elbow diagram below, we can see that the explained variance beyond 50 is negligible.

```
pca=PCA()
pca.fit(X)

plt.figure(1, figsize=(12,8))

plt.plot(pca.explained_variance_, linewidth=2)

plt.xlabel('Components')
plt.ylabel('Explained Variaces')
plt.show()
```

Figure 4.16 Code for elbow diagram

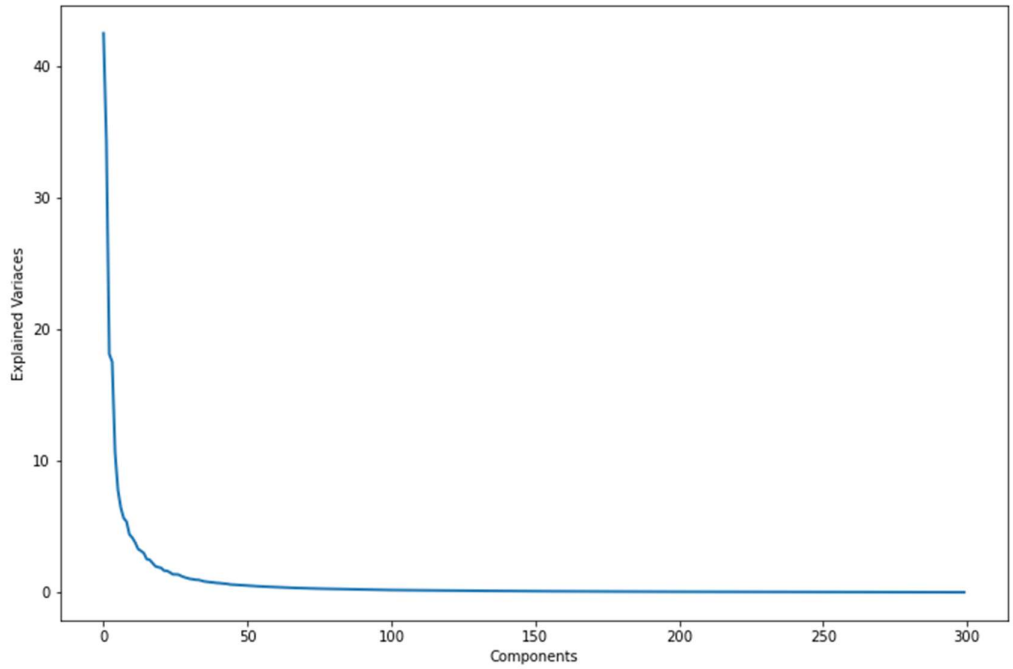

Figure 4.17 Elbow diagram of number of components and explained variances

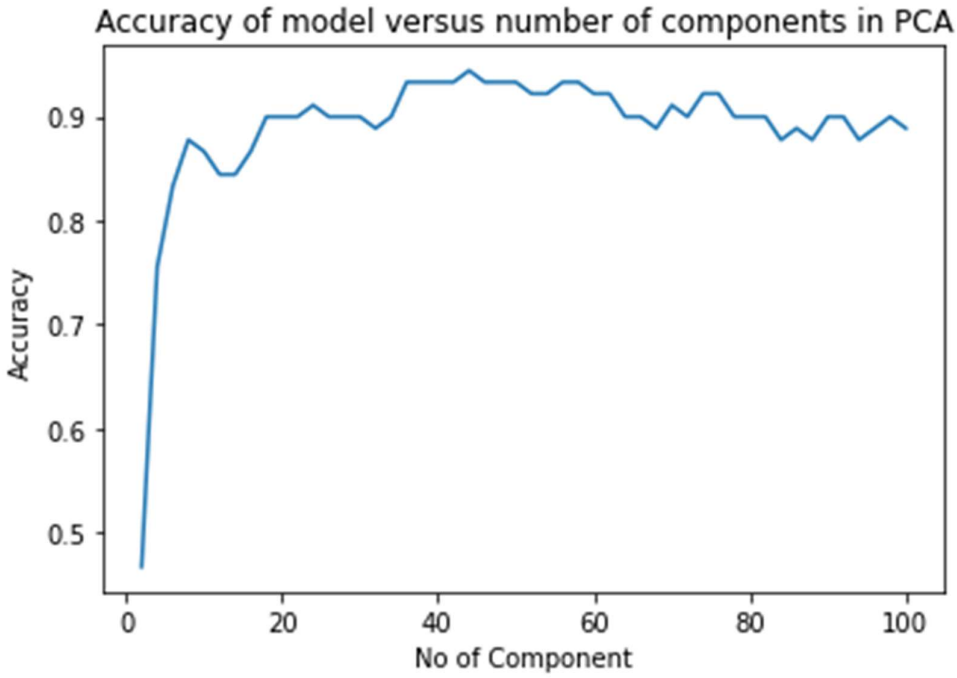

Figure 4.18 Accuracy of model versus number of components selection in PCA

However, the accuracy vs $n\_components$ graph can only be used as a rough estimation. In our experiment, we have confirmed that the proposed system has better robustness on prediction

in real-time face recognition. Setting $n\_components$ to a very small value (e.g. 20) will result in failure to predict when we perform real-time face recognition.

By observing the elbow diagram, evaluating the decision accuracy, and observing the behavior at real-time prediction, the value of $n\_components$ is chosen as 60. This means the first 60 components was selected as input to SVM for classification.

```
n_components = 52
print("Training PCA with {} components...".format(n_components))
pca = PCA(n_components=n_components, whiten=True)
pca.fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

clf = SVC(probability=True)
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print("No of components in PCA: ", n_components)
print("accuracy score:{:.4f}".format(metrics.accuracy_score(y_test, y_pred)))
input("Press enter to start real-time recognition...")
```

```
Training PCA with 52 components...
No of components in PCA:  52
accuracy score:0.9333
```

Figure 4.19 Result of training with 52 components of PCA and SVC machine learning method

5. Real-Time Face Recognition

As introduced in Section 4, in order to achieve real-time face recognition, both real-time face detection and real-time model inference (identification) are needed.

With OpenCV DNN Face Detector and Eigenface, our designed face recognition system can run at real-time on mid- and low-end CPUs.

Some advantages of our face recognition system are summarized as follows:

i) Our designed face recognition system is able to handle multiple subjects within the frame concurrently.

ii) Our designed face recognition system is able to tackle with mask conditions as well as detecting if the subjects are wearing masks. It is useful in this COVID pandemic.

iii) By thresholding the confidence level, our recognition system can handle unseen faces.

5.1. Regular Face Recognition (without Mask)

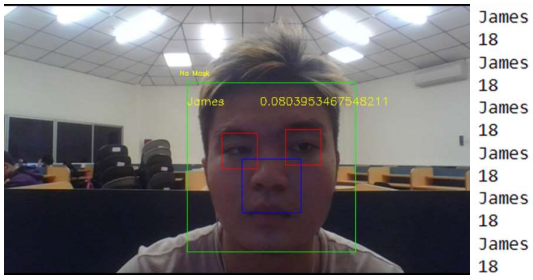

Figure 5.1 Real-time recognition and records the name and id of person – James with id 18

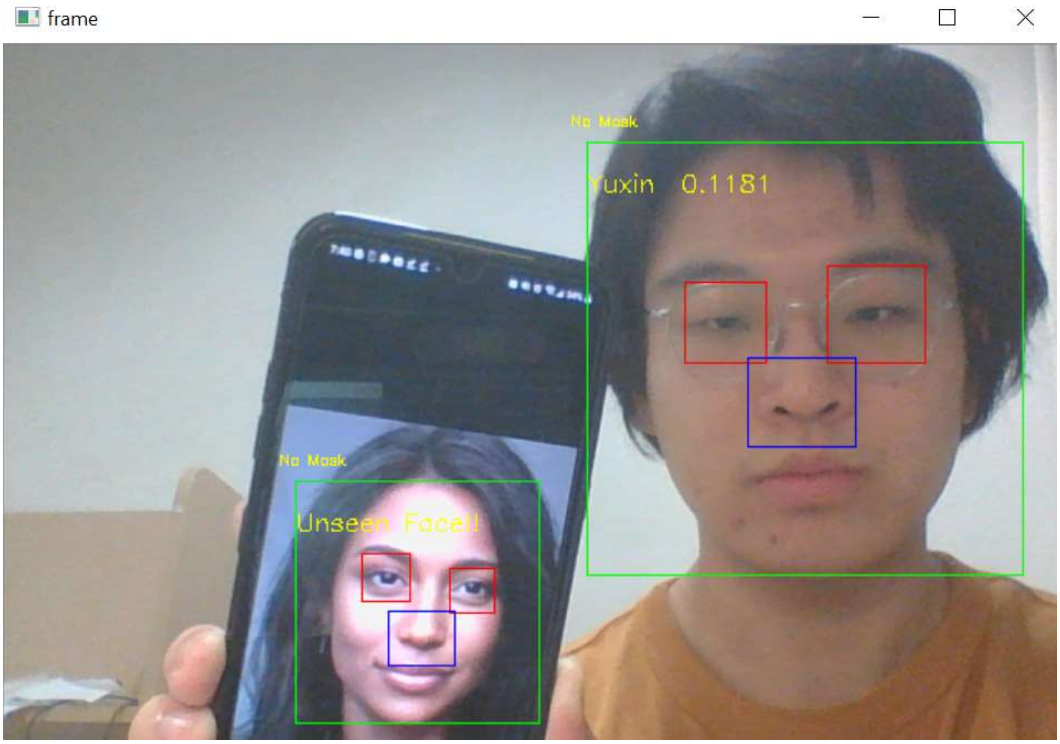

Figure 5.2 Real-time recognition without mask. The random female image is from google.

As introduced in Section 4, when a face is not partially covered by mask, it will be sent to the regular model for identification. Mask status, name, and confidence level are displayed on the top left corner.

5.1.1. Handling Unseen Faces (No Mask)

From our observation, the confidence level of unseen faces without mask are usually below 0.1, while faces in database are usually at around 0.12 to 0.13. Therefore, thresholding with thresh = 0.1 is performed to differentiate unseen faces from faces in our dataset.

5.2. Additional Task: Face Recognition with Mask

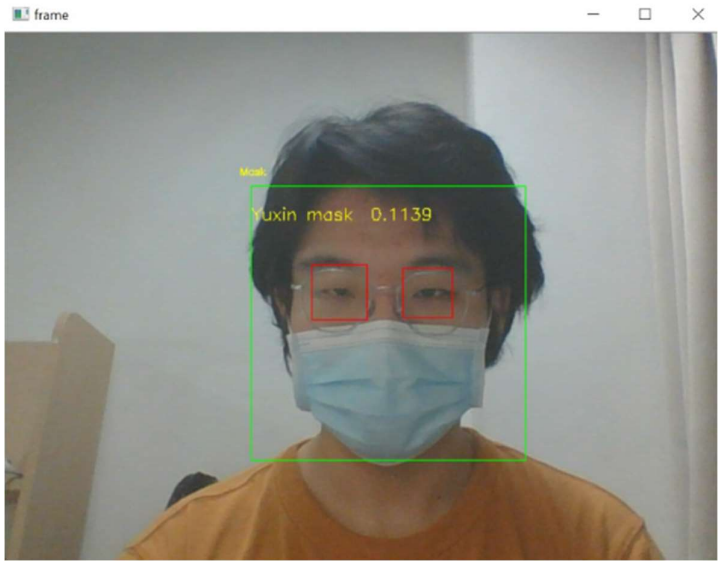

Figure 5.3 Real-time recognition when mask is present.

As shown in Figure 4.1, when a mask is present in a detected face, the upper half of the detected face will be used as input for the model for face recognition with mask. Figure 5.3

5.2.1. Handling Unseen Faces (Mask)

From our observation, the confidence level of unseen faces with mask are usually at around 0.04 to 0.07, while faces in database are at around 0.075 to 0.09. Therefore, thresholding with thresh = 0.07 is performed to differentiate unseen faces from faces in our dataset.

However, since the gap of the two thresholds are very small, the ability to handle unseen faces with mask is very limited. Furthermore, in some extreme cases, unseen faces with mask can also produce a confidence level of 0.09.

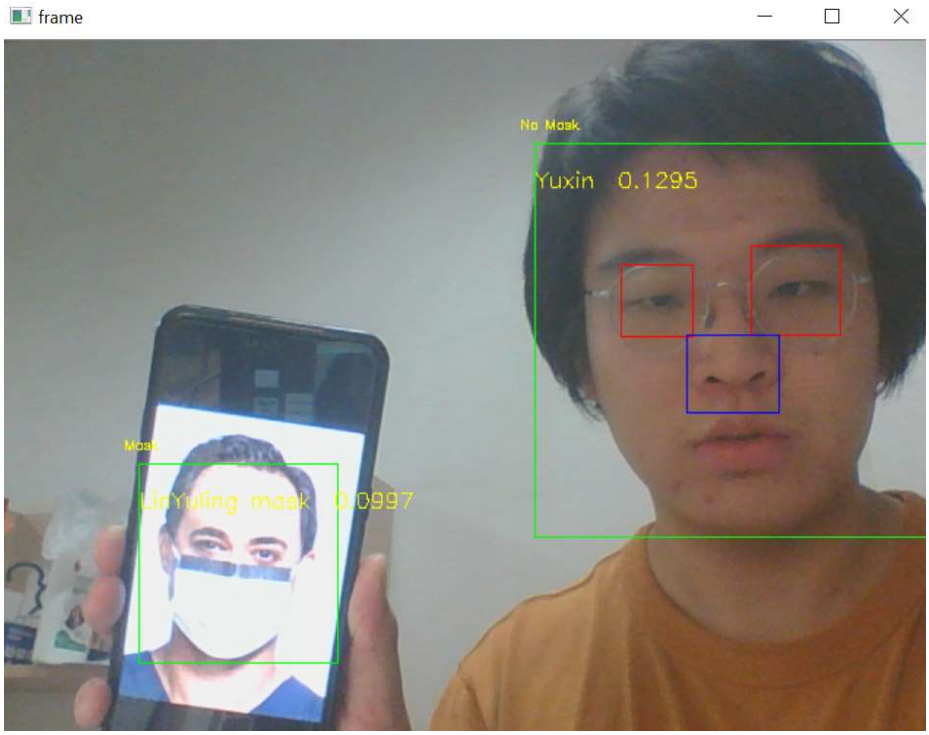

Figure 5.4 Failure to handle unseen face with mask

This result is not surprising since the occlusion is substantial. Due to the substantial occlusion, discriminative parts such as nose, mouth, chin, cheeks are all covered.

Since PCA is not robust towards covariate changes and occlusions, this is so far the best results we can obtain. When using PCA face recognition with mask, depending on the preference on yielding false positives (i.e. labelling unseen face as someone in the database when confidence level is high) or false negatives (i.e. labelling faces in the database as unseen if the confidence level is not high enough. This gives better security but more recognition failures), the threshold value should be carefully selected according to the specific task.

```
# <editor-fold desc="real-time recognition">

# -- 1. Load the cascades
if not face_cascade.load(face_cascade_name):
    print('--(!)Error loading face cascade')
    exit(0)
if not eyes_cascade.load(eyes_cascade_name):
    print('--(!)Error loading eyes cascade')
    exit(0)
if not nose_cascade.load(nose_cascade_name):
    print('--(!)Error loading nose cascade')
    exit(0)
camera_device = 0
```

```
# -- 2. Read the video stream
cap = cv.VideoCapture(camera_device)
if not cap.isOpened:
    print('--(!)Error opening video capture')
    exit(0)
while True:
    ret, frame = cap.read()
    if frame is None:
        print('--(!) No captured frame -- Break!')
        break
    # detect face, eyes and nose
    frame, coords = detectFaceDNN(frame)
    # display detected faces
    detected_faces = getDetectedFaces(frame, coords)
    # draw rectangles
    drawDetectionResults(frame, coords)
    # get mask status for all detected faces
    # list_mask_status = [(x and y) for (x, y) in
    #                     zip(getMaskStatus(frame, coords), [not elem for elem in ifExistTrueNose(coords)])]
    list_mask_status = [not elem for elem in ifExistTrueNose(coords)]

    # put mask status onto frame
    putMaskStatus(frame, coords, list_mask_status)

    # get recognition results
    ids = getPCAResults(detected_faces)

    # put on frame
    putIdentity(frame, coords, ids)

    cv.imshow("frame", frame)
    # avoid fake nose at eyebrow
    # print()
    if cv.waitKey(10) == 27:
        break
```

Figure 5.5 Code for real-time face recognition

6. Conclusion

In this project, by introducing OpenCV DNN Face Detector, Haar Cascade Detector (nose, eyes) and SVM along with Eigenface, we have developed a PCA-based real-time face recognition system. Furthermore, we have also made improvements for the system to handle face masks and unseen faces. Although Eigenface has limited robustness towards covariate changes and occlusions, due to its low computational cost and easiness to implement, it is still an effective algorithm in practical use, and a great algorithm to get hands-on practice.

7. Further work recommendation and PCA pros and cons in face recognition

7.1. Further Work Recommendations

Currently, our face detection model can already achieve the detection of face id information in real time and will not be affected by wearing masks under the Covid19 situation. Our model can even help detect whether the target person is wearing a mask as required and make a reminder. Moreover, our face detection model can also support multi-person scenarios. However, our face detection model still has many places that can be further improved, such as the ability to recognize emotional expressions of faces, the ability to support real-time face detection in fast-moving scenes, and the ability to display and record real-time monitoring time under the needs of some work supervision scenarios.

7.2. Advantages of PCA Method in Face Recognition

7.2.1. Delete correlated features

In real-world scenarios, it is quite common to have thousands of features in a used dataset. However, it is impossible to visualize so many features in one graph which makes analysis hard to implement. Manually finding correlations among thousands of features is almost unachievable, time-consuming and frustrating. Therefore, it is necessary to reduce the number of features in the used dataset in some way. Fortunately, PCA solves all these problems. The independent nature of all principal components in PCA method means that there is no correlation between them anymore.

7.2.2. Improve Algorithm Performance

We cannot utilize all features in our algorithm because it will absolutely slow down the performance of our algorithm. PCA is an effective method which speeds up algorithms through eliminating correlated variables that do not contribute to any decision making. Therefore, the training time of the algorithm drops down dramatically due to the decrease of the number of features.

7.2.3. Reduce Overfitting

The occurrence of overfitting is usually because of the substantial number of variables in the dataset. Therefore, PCA helps overcome the overfitting problem through the reduction of features.

7.2.4. Improve Visualization

High-dimensional data is difficult to visualize and understand. PCA transforms high-dimensional data into low-dimensional data, usually two-dimensional. Therefore, we can then draw a 2D Plot to analysis which principal components cause higher variance or have a larger impact than others. The PCA method thus makes visualization easier.

7.3. Disadvantages of PCA Method in Face Recognition

7.3.1. Independent variables become difficult to interpret

Original features will become principal components after implementing PCA on the dataset. The principal component is a linear combination of the original features. Therefore, it is unavoidable that principal components are less interpretable or readable than the original features.

7.3.2. Data normalization must be done before PCA

Normalization of the data must be done before implementing PCA method, otherwise the best principal components will not be able to be found by PCA. For example, if the data of a feature set is represented in millions, the large variance scale of the training set will lead to large load of features. Therefore, the principal components will be biased towards high-variance features, resulting in erroneous results.

7.3.3. Loss of information

Even though PCA method is always trying to make principal components cover the maximum variance between features in the dataset, if we do not choose the number of principal components carefully, some information may still lose in comparison with the original features.

References

[1] A. E.-B. A. A. E.-H. Ahmad Tolba, "Face Recognition: A Literature Review," *International Journal of Signal Processing ER,* pp. 88-103, 01 01 2005.

[2] K. P. F.R.S., "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science,* vol. 2, pp. 559-572, 1901.

[3] M. a. P. A. Turk, "Face recognition using eigenfaces," in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1991.

Appendix

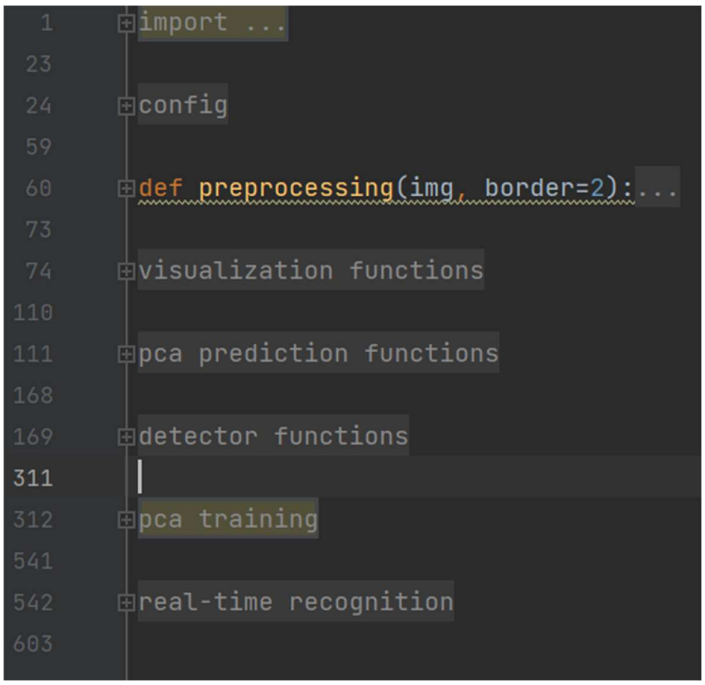

Figure 0.1 Structure of the python script.

The whole code is available in the folder of PCA Face Recognition submitted.