INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY

Valiamala P.O., Thiruvananthapuram - 695 547 Kerala, India.

Face Detection and Recognition

ESG 624 – Pattern Recognition and Machine Learning

Project Report

Developed By-

CHAITANYA JOSHI

SC19M001

M.Tech, Geoinformatics (2019-21)

Guided By-

Dr. DEEPAK MISHRA

Table of Contents

List of Figures

Introduction -

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly.

Face Detection -

Face Detection is the first and essential step for face recognition, and it is used to detect faces in the images. It is a part of object detection and can be used in many areas such as security, bio-metrics, law enforcement, entertainment, personal safety, etc.

It is used to detect faces in real time for surveillance and tracking of persons or objects. It is widely used in cameras to identify multiple appearances in the frame Ex- Mobile cameras and DSLRs. Facebook is also using face detection algorithms to detect faces in the images and recognise them.

Face Detection Methods -

1. Feature Based
2. Appearance Based
    a. Eigenface Based
    b. Distribution Based
    c. Neural Networks
    d. SVM
    e. Naive Bayes Classifiers
    f. Hidden Markov Models
3. Knowledge Based
4. Template Matching

The report has 3 parts -

1. Eigenface Generation using PCA.
2. Detection and Recognition of Faces (Live feed using webcam).
3. Face Recognition Attendance System.

Part I - Eigenface Generation using PCA

Face recognition is ubiquitous in science fiction: the protagonist looks at a camera, and the camera scans his or her face to recognize the person. More formally, we can formulate face recognition as a classification task, where the inputs are images and the outputs are people's names. We're going to discuss a popular technique for face recognition called eigenfaces. And at the heart of eigenfaces is an unsupervised dimensionality reduction technique called principal component analysis (PCA), and we will see how we can apply this general technique to our specific task of face recognition.

A set of eigenfaces can be generated by performing a mathematical process called principal component analysis (PCA) on a large set of images depicting different human faces.Informally, eigenfaces can be considered a set of "standardized face ingredients", derived from statistical analysis of many pictures of faces.

Any human face can be considered to be a combination of these standard faces. For example, one's face might be composed of the average face plus 10% from eigenface 1, 55% from eigenface 2, and even −3% from eigenface 3. Remarkably, it does not take many eigenfaces combined together to achieve a fair approximation of most faces. Also, because a person's face is not recorded by a digital photograph, but instead as just a list of values (one value for each eigenface in the database used), much less space is taken for each person's face.

The eigenfaces that are created will appear as light and dark areas that are arranged in a specific pattern. This pattern is how different features of a face are singled out to be evaluated and scored. There will be a pattern to evaluate symmetry, whether there is any style of facial hair, where the hairline is, or an evaluation of the size of the nose or mouth. Other eigenfaces have patterns that are less simple to identify, and the image of the eigenface may look very little like a face.

The technique used in creating eigenfaces and using them for recognition is also used outside of face recognition: handwriting recognition, lip reading, voice recognition, sign language/hand gestures interpretation and medical imaging analysis. Therefore, some do not use the term eigenface, but prefer to use 'eigenimage'.

Implementation -

- Language Used - Python 3.7
- Input Data Format - CSV File

Code -

```
import matplotlib.pyplot as plt
from time import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from sklearn.svm import SVC
import numpy as np
import pandas as pd

# Helper functions. Use when needed.
def show_orignal_images(pixels):
    # Displaying Original Images
    fig, axes = plt.subplots(6, 10, figsize=(11, 7),
                             subplot_kw={'xticks': [], 'yticks': []})
    for i, ax in enumerate(axes.flat):
        ax.imshow(np.array(pixels)[i].reshape(64, 64), cmap='gray')
    plt.show()

def show_eigenfaces(pca):
    # Displaying Eigenfaces
    fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                             subplot_kw={'xticks': [], 'yticks': []})
    for i, ax in enumerate(axes.flat):
        ax.imshow(pca.components_[i].reshape(64, 64), cmap='gray')
        ax.set_title("PC " + str(i + 1))
    plt.show()

# Step 1: Read dataset and visualize it
df = pd.read_csv("face_data.csv")
targets = df["target"]
pixels = df.drop(["target"], axis=1)

print(np.array(pixels).shape)
```

```
show_orignal_images(pixels)
# Step 2: Split Dataset into training and testing
x_train, x_test, y_train, y_test = train_test_split(pixels, targets)

# Step 3: Perform PCA.
pca = PCA(n_components=150).fit(x_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
plt.show()

show_eigenfaces(pca)

# Step 4: Project Training data to PCA
print("Projecting the input data on the eigenfaces orthonormal basis")
Xtrain_pca = pca.transform(x_train)

# Step 5: Initialize Classifier and fit training data
clf = SVC(kernel='rbf', C=1000, gamma=0.001)
clf = clf.fit(Xtrain_pca, y_train)

# Step 6: Perform testing and get classification report
print("Predicting people's names on the test set")
t0 = time()
Xtest_pca = pca.transform(x_test)
y_pred = clf.predict(Xtest_pca)
print("done in %0.3fs" % (time() - t0))
print(classification_report(y_test, y_pred))
```

Results -

The input data had 40 faces and the resultant eigenfaces are 24.

Accuracy Report Generated in Python IDE using the classification_report function of sklearn -

```
Projecting the input data on the eigenfaces orthonormal basis
Predicting people's names on the test set done in 0.020s
```

```
    precision   recall  f1-score   support

 0      1.00      1.00      1.00         3
 1      1.00      1.00      1.00         2
 2      1.00      0.40      0.57         5
 3      1.00      1.00      1.00         3
 4      1.00      1.00      1.00         2
 5      1.00      1.00      1.00         2
 6      1.00      1.00      1.00         3
 7      1.00      1.00      1.00         1
 8      1.00      1.00      1.00         6
 9      1.00      1.00      1.00         2
10      1.00      1.00      1.00         1
11      1.00      1.00      1.00         4
12      1.00      1.00      1.00         2
13      1.00      1.00      1.00         1
14      1.00      1.00      1.00         2
15      1.00      1.00      1.00         4
16      1.00      1.00      1.00         1
17      1.00      1.00      1.00         4
19      1.00      1.00      1.00         2
20      0.86      1.00      0.92         6
21      1.00      0.67      0.80         3
22      0.67      1.00      0.80         2
23      1.00      1.00      1.00         1
24      1.00      1.00      1.00         1
25      1.00      0.67      0.80         3
26      1.00      1.00      1.00         2
27      1.00      1.00      1.00         2
28      1.00      1.00      1.00         3
29      1.00      1.00      1.00         2
30      1.00      1.00      1.00         2
31      1.00      1.00      1.00         5
32      1.00      1.00      1.00         2
34      1.00      1.00      1.00         1
35      1.00      1.00      1.00         4
```

36	1.00	1.00	1.00	4
37	1.00	1.00	1.00	4
38	0.50	1.00	0.67	1
39	0.50	1.00	0.67	2
accuracy			0.95	100
macro avg	0.96	0.97	0.95	100
weighted avg	0.97	0.95	0.95	100

Original Images -

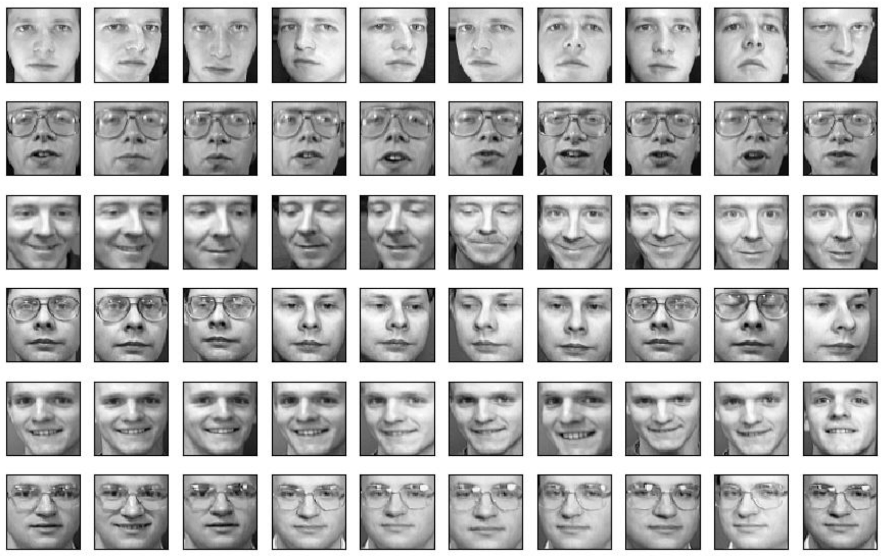


Fig 1. Original Faces from the CSV

Variance vs Number of Components Plot -

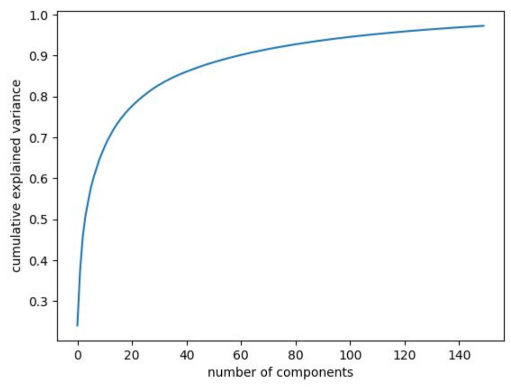


Fig 2. Plot of Variance vs Number of Components

Eigen Faces -

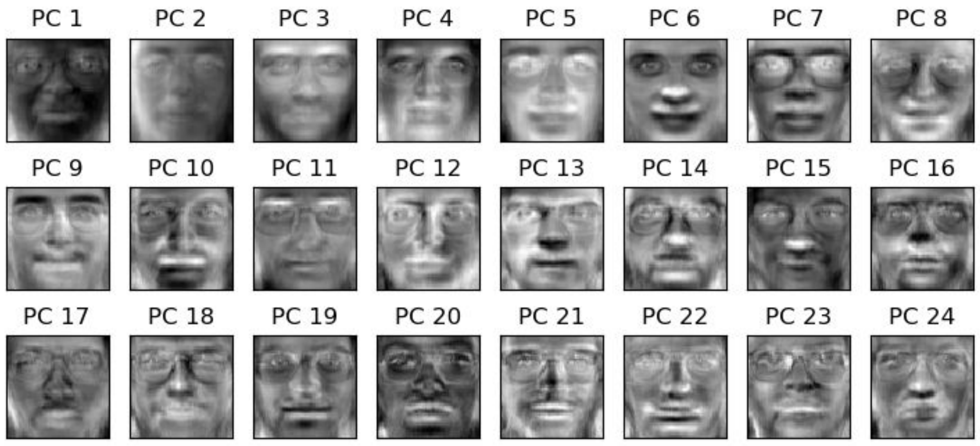


Fig 3. Generated Eigenfaces

Inference -

Each image in the original dataset has 4096 features, which using the concept of Principal Component Analysis is reduced to around 150 features and still we are able to get an accuracy close to 98% as shown in the plot (Fig 2).

Part II - Detection and Recognition of Faces

Here, we make use of the Haar Cascade implementation in the Python Programming language and multiple other open source libraries described in the sections (Part III) to be followed.

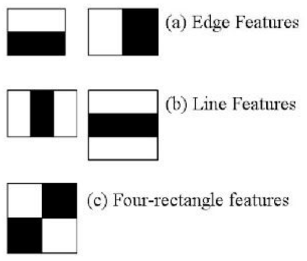
Haar Cascade -

It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

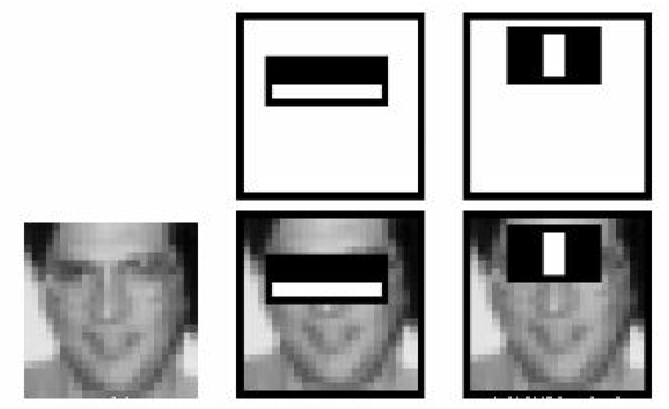
The algorithm has four stages:

1. Haar Feature Selection
2. Creating Integral Images
3. Adaboost Training
4. Cascading Classifiers

- Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001.
- It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.
- Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier.
- Then we need to extract features from it. For this, haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting the sum of pixels under white rectangle from the sum of pixels under black rectangle.



- Now all possible sizes and locations of each kernel is used to calculate plenty of features. For each feature calculation, we need to find the sum of pixels under white and black rectangles. To solve this, they introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels.
- But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.

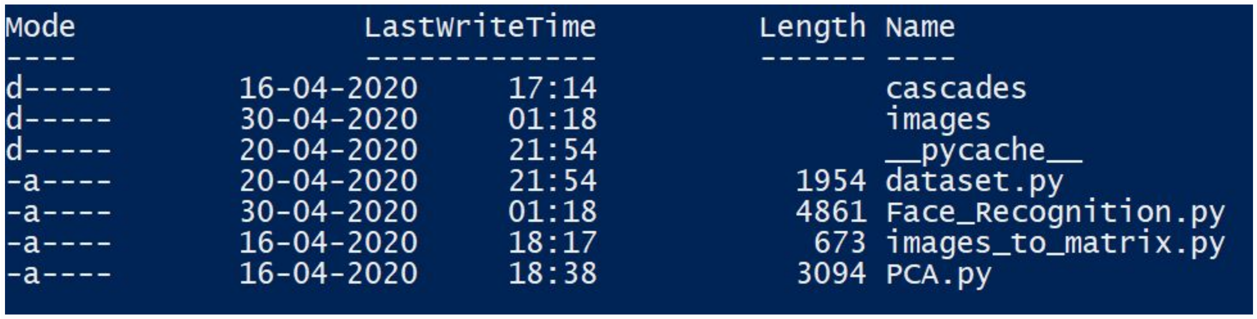


- For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with the minimum error rate, which means they are the features that best classify the face and non-face images.
- Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features.
- In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. Instead focus on the region where there can be a face. This way, we can find more time to check a possible face region.
- For this we use **Cascade of Classifiers**. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the

second stage of features and continue the process. The window which passes all stages is a face region.

Implementation -

Directory Structure:



```
Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        16-04-2020     17:14                cascades
d-----        30-04-2020     01:18                images
d-----        20-04-2020     21:54                __pycache__
-a----        20-04-2020     21:54           1954 dataset.py
-a----        30-04-2020     01:18           4861 Face_Recognition.py
-a----        16-04-2020     18:17            673 images_to_matrix.py
-a----        16-04-2020     18:38           3094 PCA.py
```

Fig 4. Directory Structure of Part II

- The code has 4 python files-
  - dataset.py
  - Face_Recognition.py
  - Images_to_matrix.py
  - PCA.py

- There 2 folders other then the 4 files-
  - cascades - Has the haarcascade_frontalface_alt2.xml file.
  - Images - Has 38 folders, each folder has 10 facial images of a person.

Code -

i) dataset.py -

```
import os

class DatasetClass:

    def __init__(self, required_no):

        # Dataset Name
        self.dir =
("C:/Users/Chaitanya/Desktop/ML_Project_Submit/PCA_FD_Project/FaceRecognitionUsingPCA/images/
ORL")
```

```
        self.images_name_for_train = []
        self.target_name_as_array = []
        self.target_name_as_set = {}
        self.y_for_train = []
        self.no_of_elements_for_train = []

        self.images_name_for_test = []
        self.y_for_test = []
        self.no_of_elements_for_test = []

        per_no = 0
        for name in os.listdir(self.dir):
            dir_path = os.path.join(self.dir, name)
            if os.path.isdir(dir_path):
                if len(os.listdir(dir_path)) >= required_no:
                    i = 0
                    for img_name in os.listdir(dir_path):
                        img_path = os.path.join(dir_path, img_name)

                        if i < required_no:
                            self.images_name_for_train += [img_path]
                            self.y_for_train += [per_no]
                            if len(self.no_of_elements_for_train) > per_no:
                                self.no_of_elements_for_train[per_no] += 1
                            else:
                                self.no_of_elements_for_train += [1]

                            if i == 0:
                                self.target_name_as_array += [name]
                                self.target_name_as_set[per_no] = name

                        else:
                            self.images_name_for_test += [img_path]
                            self.y_for_test += [per_no]
                            if len(self.no_of_elements_for_test) > per_no:
```

```
                        self.no_of_elements_for_test[per_no] += 1
                    else:
                        self.no_of_elements_for_test += [1]

                i += 1
            per_no += 1
```

ii) Face_Recognition.py -

```
import cv2
import time

# importing algorithms
from PCA import PcaClass

# importing feature extraction classes
from images_to_matrix import ImagesToMatrixClass
from dataset import DatasetClass

# for video = 1
reco_type = 1

# No of images For Training(Left will be used as testing Image)
no_of_images_of_one_person = 8
dataset_obj = DatasetClass(no_of_images_of_one_person)

# Data For Training
images_names = dataset_obj.images_name_for_train
y = dataset_obj.y_for_train
no_of_elements = dataset_obj.no_of_elements_for_train
target_names = dataset_obj.target_name_as_array

# Data For Testing
images_names_for_test = dataset_obj.images_name_for_test
y_for_test = dataset_obj.y_for_test
no_of_elements_for_test = dataset_obj.no_of_elements_for_test
```

```
training_start_time = time.process_time()
img_width, img_height = 50, 50

i_t_m_c = ImagesToMatrixClass(images_names, img_width, img_height)

scaled_face = i_t_m_c.get_matrix()
# cv2.imshow("Original Image" , cv2.resize(np.array(np.reshape(scaled_face[:,1],[img_height,
img_width]), dtype = np.uint8),(200, 200)))
# cv2.waitKey()

# Algo

my_algo = PcaClass(scaled_face, y, target_names, no_of_elements, 90)

new_coordinates = my_algo.reduce_dim()
my_algo.show_eigen_face(img_width, img_height, 50, 150, 0)

training_time = time.process_time() - training_start_time

if reco_type == 0:
  time_start = time.process_time()

  correct = 0
  wrong = 0
  i = 0
  net_time_of_reco = 0

  for img_path in images_names_for_test:

    time_start = time.process_time()
    find_name = my_algo.recognize_face(my_algo.new_cord(img_path, img_height, img_width))
    time_elapsed = (time.process_time() - time_start)
    net_time_of_reco += time_elapsed
    rec_y = y_for_test[i]
    rec_name = target_names[rec_y]
    if find_name is rec_name:
          correct += 1
          print("Correct", " Name:", find_name)
    else:
          wrong += 1
          print("Wrong:", " Real Name:", rec_name, "Rec Y:", rec_y, "Find Name:", find_name)
    i += 1

  print("Correct", correct)
  print("Wrong", wrong)
  print("Total Test Images", i)
  print("Percent", correct / i * 100)
  print("Total Person", len(target_names))
  print("Total Train Images", no_of_images_of_one_person * len(target_names))
  print("Total Time Taken for reco:", time_elapsed)
  print("Time Taken for one reco:", time_elapsed / i)
  print("Training Time", training_time)

if reco_type == 1:
  face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt2.xml')

  cap = cv2.VideoCapture(0)

  while True:
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.5, minNeighbors=7)

    i = 0
    for (x, y, w, h) in faces:
          roi_gray = gray[y:y + h, x:x + w]
          scaled = cv2.resize(roi_gray, (img_height, img_width))
          rec_color = (255, 0, 0)
          rec_stroke = 2
          cv2.rectangle(frame, (x, y), (x + w, y + h), rec_color, rec_stroke)
          new_cord = my_algo.new_cord_for_image(scaled)
          name = my_algo.recognize_face(new_cord)
          font = cv2.FONT_HERSHEY_SIMPLEX
          font_color = (255, 255, 255)
          font_stroke = 2
          cv2.putText(frame, name, (x, y), font, 1, font_color, font_stroke, cv2.LINE_AA)
          i += 1
    cv2.imshow('Colored Frame', frame)
    if cv2.waitKey(20) & 0xFF == ord('q'):
          break

  cap.release()
  cv2.destroyAllWindows()
```

iii) images_to_matrix.py -

```
import cv2
import numpy as np

class ImagesToMatrixClass:

    def __init__(self, images_name, img_width, img_height):
        self.images_name = images_name
        self.img_width = img_width
        self.img_height = img_height
        self.img_size = (img_width * img_height)

    def get_matrix(self):
        col = len(self.images_name)
        img_mat = np.zeros((self.img_size, col))

        i = 0
        for name in self.images_name:
            gray = cv2.imread(name, 0)
            gray = cv2.resize(gray, (self.img_height, self.img_width))
            mat = np.asmatrix(gray)
            img_mat[:, i] = mat.ravel()
            i += 1
        return img_mat
```

iv) PCA -

```
import numpy as np
import cv2
import scipy.linalg as s_linalg


class PcaClass:

    def give_p(self, d):
        sum = np.sum(d)
        sum_85 = self.quality_percent * sum / 100
        temp = 0
        p = 0
        while temp < sum_85:
            temp += d[p]
            p += 1
        return p

    def reduce_dim(self):

        p, d, q = s_linalg.svd(self.images, full_matrices=True)
        p_matrix = np.matrix(p)
        d_diag = np.diag(d)
        q_matrix = np.matrix(q)
        p = self.give_p(d)
        self.new_bases = p_matrix[:, 0:p]
        self.new_coordinates = np.dot(self.new_bases.T, self.images)
        return self.new_coordinates.T

    def __init__(self, images, y, target_names, no_of_elements, quality_percent):
        self.no_of_elements = no_of_elements
        self.images = np.asarray(images)
        self.y = y
        self.target_names = target_names
        mean = np.mean(self.images, 1)
        self.mean_face = np.asmatrix(mean).T
        self.images = self.images - self.mean_face
        self.quality_percent = quality_percent

    def original_data(self, new_coordinates):
        return self.mean_face + (np.dot(self.new_bases, new_coordinates.T))

    def show_eigen_face(self, height, width, min_pix_int, max_pix_int, eig_no):
        ev = self.new_bases[:, eig_no:eig_no + 1]
        min_orig = np.min(ev)
        max_orig = np.max(ev)
        ev = min_pix_int + (((max_pix_int - min_pix_int) / (max_orig - min_orig)) * ev)
        ev_re = np.reshape(ev, (height, width))
        # cv2.imshow("Eigen Face " + str(eig_no),  cv2.resize(np.array(ev_re, dtype = np.uint8),(200, 200)))
        # cv2.waitKey()

    def new_cord(self, name, img_height, img_width):
        img = cv2.imread(name)
        gray = cv2.resize(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), (img_height, img_width))
        img_vec = np.asmatrix(gray).ravel()
        img_vec = img_vec.T
        new_mean = ((self.mean_face * len(self.y)) + img_vec) / (len(self.y) + 1)
        img_vec = img_vec - new_mean
        return np.dot(self.new_bases.T, img_vec)

    def new_cord_for_image(self, image):
        img_vec = np.asmatrix(image).ravel()
        img_vec = img_vec.T
        new_mean = ((self.mean_face * len(self.y)) + img_vec) / (len(self.y) + 1)
        img_vec = img_vec - new_mean
        return np.dot(self.new_bases.T, img_vec)

    def recognize_face(self, new_cord_pca, k=0):
```

```
    classes = len(self.no_of_elements)
    start = 0
    distances = []
    for i in range(classes):
        temp_imgs = self.new_coordinates[:, int(start): int(start + self.no_of_elements[i])]
        mean_temp = np.mean(temp_imgs, 1)
        start = start + self.no_of_elements[i]
        dist = np.linalg.norm(new_cord_pca - mean_temp)
        distances += [dist]
    min = np.argmin(distances)

    # Temp Threshold
    threshold = 100000
    if distances[min] < threshold:
        print(self.target_names[min])
        return self.target_names[min]
    else:
        print(min, 'Unknown')
        return 'Unknown'
```

Results -

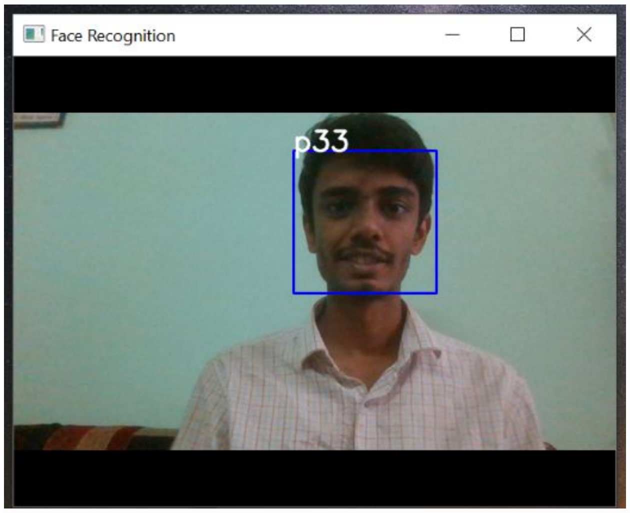
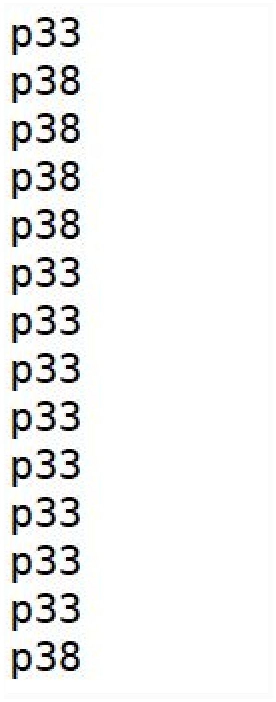


Fig 5. Face Recognized using Webcam



```
p33
p38
p38
p38
p38
p33
p33
p33
p33
p33
p33
p33
p33
p38
```

Fig 6. Predicted face for each frame (last 14 frames)

Accuracy :

Reloaded modules: PCA, images_to_matrix, dataset

```
p1
Correct  Name: p1
p1
Correct  Name: p1
p10
Correct  Name: p10
p10
Correct  Name: p10
p11
Correct  Name: p11
p11
Correct  Name: p11
p12
Correct  Name: p12
p12
Correct  Name: p12
p13
```

```
Correct  Name: p13
p13
Correct  Name: p13
p14
Correct  Name: p14
p14
Correct  Name: p14
p25
Wrong:  Real Name: p15 Rec Y: 6 Find Name: p25
p15
Correct  Name: p15
p16
Correct  Name: p16
p16
Correct  Name: p16
p17
Correct  Name: p17
p17
Correct  Name: p17
p18
Correct  Name: p18
p18
Correct  Name: p18
p19
Correct  Name: p19
p19
Correct  Name: p19
p2
Correct  Name: p2
p2
Correct  Name: p2
p20
Correct  Name: p20
p20
Correct  Name: p20
p21
```

```
Correct  Name: p21
p21
Correct  Name: p21
p22
Correct  Name: p22
p22
Correct  Name: p22
p23
Correct  Name: p23
p23
Correct  Name: p23
p24
Correct  Name: p24
p24
Correct  Name: p24
p25
Correct  Name: p25
p25
Correct  Name: p25
p35
Wrong:  Real Name: p26 Rec Y: 18 Find Name: p35
p26
Correct  Name: p26
p27
Correct  Name: p27
p38
Wrong:  Real Name: p27 Rec Y: 19 Find Name: p38
p28
Correct  Name: p28
p28
Correct  Name: p28
p10
Wrong:  Real Name: p29 Rec Y: 21 Find Name: p10
p20
Wrong:  Real Name: p29 Rec Y: 21 Find Name: p20
p3
```

```
Correct  Name: p3
p3
Correct  Name: p3
p30
Correct  Name: p30
p30
Correct  Name: p30
p31
Correct  Name: p31
p31
Correct  Name: p31
p32
Correct  Name: p32
p32
Correct  Name: p32
p33
Correct  Name: p33
p33
Correct  Name: p33
p34
Correct  Name: p34
p34
Correct  Name: p34
p35
Correct  Name: p35
p35
Correct  Name: p35
p36
Correct  Name: p36
p36
Correct  Name: p36
p37
Correct  Name: p37
p37
Correct  Name: p37
p38
```

```
Correct  Name: p38
p38
Correct  Name: p38
p4
Correct  Name: p4
p4
Correct  Name: p4
p5
Correct  Name: p5
p5
Correct  Name: p5
p6
Correct  Name: p6
p6
Correct  Name: p6
p7
Correct  Name: p7
p7
Correct  Name: p7
p8
Correct  Name: p8
p8
Correct  Name: p8
p9
Correct  Name: p9
p9
Correct  Name: p9
Correct 71
Wrong 5
Total Test Images 76
Percent 93.42105263157895
Total Person 38
Total Train Images 304
Total Time Taken for reco: 0.0
Time Taken for one reco: 0.0
Training Time 1.1875
```

Inference -

This implementation has a fairly good accuracy of 90%+ every time but also is dependent on the following conditions:

1. The lighting(illumination) has to be optimal, the recognition performs poorly under low lighting conditions.
2. The results are affected if the user is having spectacles and hence the screen of the device (laptop here) is getting reflected in the glasses which is leading to the hindrance of the eyes.

Here we have compared the input feed from the webcam with the dataset we had in our system from which we have derived the eigenfaces.

The next part is the demonstration of an end to end software of Face Recognition Attendance System.

Part III - Face Recognition Attendance System

In this project we've used the supervised machine learning technique to implement the Face Recognition Based Attendance System.

The project makes use of the Tkinter library for the development of graphical user interface (GUI).

Supervised Machine Learning -

The majority of practical machine learning uses supervised learning.

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

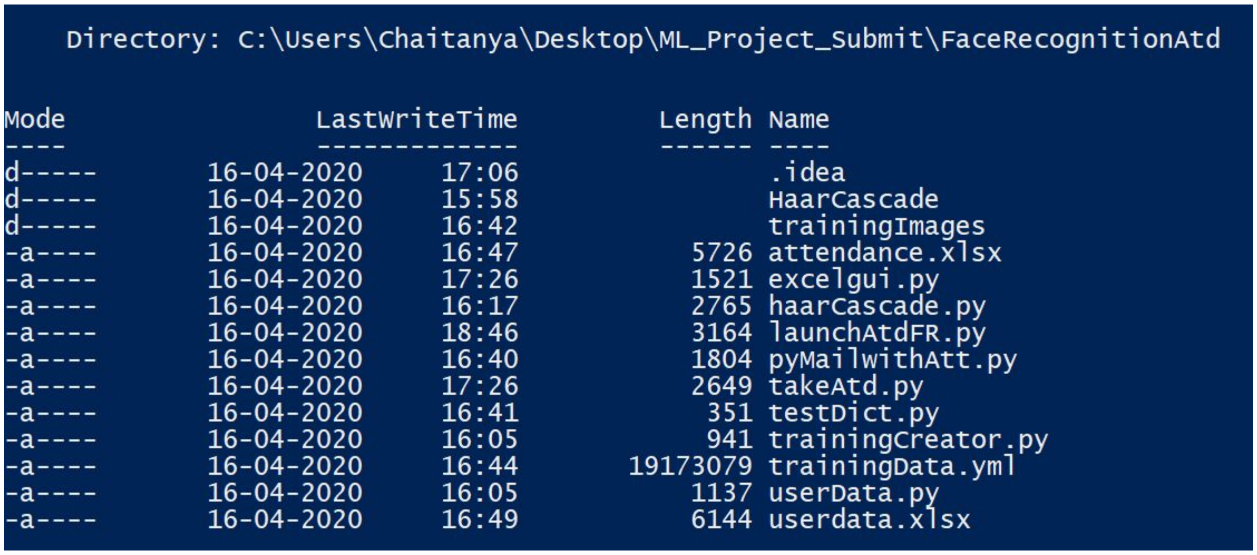
It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Libraries Used -

- **cv2 -** OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.
    - OpenCV Python - https://opencv-python-tutroals.readthedocs.io/en/latest/index.html
    - OpenCV - https://opencv.org/
- **NumPy -** NumPy is the fundamental package for scientific computing with Python. It contains among other things:
    - a powerful N-dimensional array object
    - sophisticated (broadcasting) functions
    - tools for integrating C/C++ and Fortran code

    - useful linear algebra, Fourier transform, and random number capabilities
    - https://numpy.org/
- **Openpyxl -** openpyxl is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files.
    - https://openpyxl.readthedocs.io/en/stable/
- **Xl2dict -** xl2dict is a python module to convert spreadsheets into a python dictionary. The input is a spreadsheet (xls or xlsx) and the output is a list of dictionaries.
    - https://pypi.org/project/xl2dict/
- **Pandas -** pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
    - https://pandas.pydata.org/
- **Datetime -** The datetime module supplies classes for manipulating dates and times.
    - https://docs.python.org/3/library/datetime.html
- **Smtplib -** The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.
    - https://docs.python.org/3/library/smtplib.html
- **Email** - The email package is a library for managing email messages.
    - https://docs.python.org/3/library/email.html
- **Tkinter -** The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, as well as on Windows systems.
    - https://docs.python.org/3/library/tkinter.html

Implementation -



```
    Directory: C:\Users\Chaitanya\Desktop\ML_Project_Submit\FaceRecognitionAtd


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        16-04-2020     17:06                .idea
d-----        16-04-2020     15:58                HaarCascade
d-----        16-04-2020     16:42                trainingImages
-a----        16-04-2020     16:47           5726 attendance.xlsx
-a----        16-04-2020     17:26           1521 excelgui.py
-a----        16-04-2020     16:17           2765 haarCascade.py
-a----        16-04-2020     18:46           3164 launchAtdFR.py
-a----        16-04-2020     16:40           1804 pyMailwithAtt.py
-a----        16-04-2020     17:26           2649 takeAtd.py
-a----        16-04-2020     16:41            351 testDict.py
-a----        16-04-2020     16:05            941 trainingCreator.py
-a----        16-04-2020     16:44       19173079 trainingData.yml
-a----        16-04-2020     16:05           1137 userData.py
-a----        16-04-2020     16:49           6144 userdata.xlsx
```

Fig 7. Directory Structure for Part III (Face Recognition Based Attendance System)

- The directory has 2 folders, 8 Python Scripts, 1 YAML file and 2 excel sheets.
- Folders:
    - HaarCascade - It has the haarcascade_frontalface_default.xml file used for face detection.
    - trainingImages - The images used as training data are stored in this folder.
- Python Scripts:
    - launchAtdFR.py - This script opens up the home window (GUI) of the application developed in tkinter.
    - trainingCreator.py - Captures images through the laptop's webcam and creates the training dataset.
    - haarCascade.py - Used to detect the face of the user.
    - excelgui.py - Launches a form GUI for user input in order to store information regarding the face captured for training.
    - userData.py - Takes in textual data corresponding to the images captured by the trainingCreator.py and stores in an excel sheet (userdata.xlsx).
    - testDict.py - Converts the data stored in excel sheet (userdata.xlsx) created by userData.py to a dictionary.
    - takeAtd.py - Takes attendance using the Face Recognition technique and makes use of the training data captured earlier to label the face. Stores the output to excel sheet (attendance.xlsx).
    - pyMailwithAtt.py - Used to send email of the attendance.
- YAML File:
    - trainingData.yml - YAML is a data transfer language like JSON which stores the data in key - value pairs. This file holds the features of the images stored in the trainingImages folder and assigns a label to it.
- Excel Sheets:
    - userdata.xlsx - Stores the Name, Branch/Department and SC code of every student in the trainingImages folder.
    - attendance.xlsx - Stores the attendance for the corresponding data.

Home Screen -

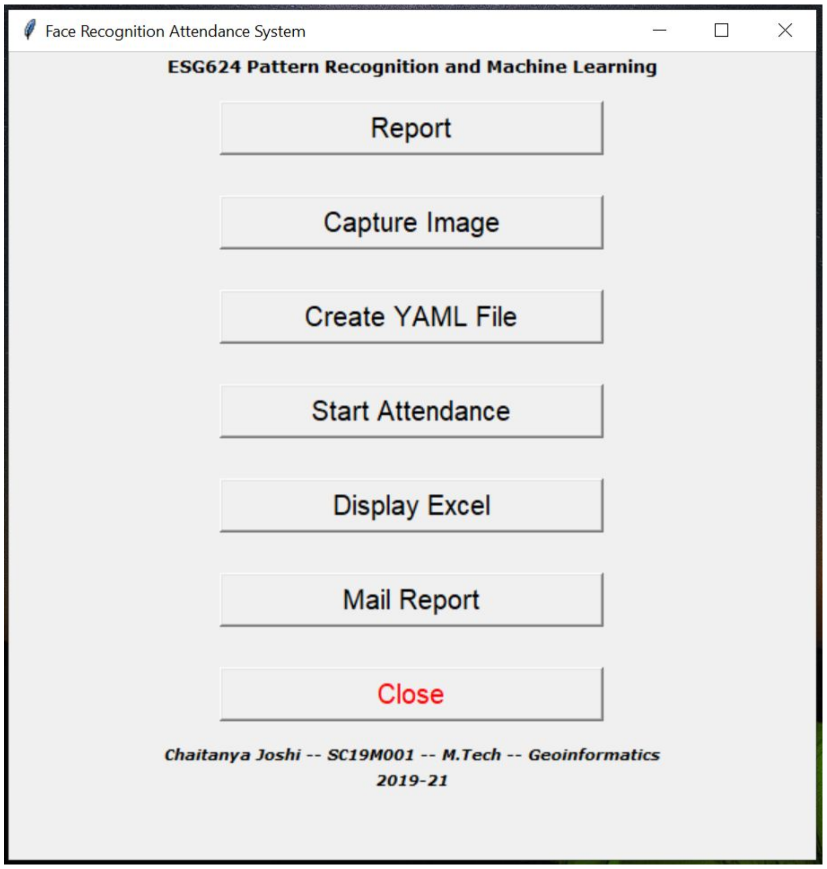

Face Recognition Attendance System
ESG624 Pattern Recognition and Machine Learning
Report
Capture Image
Create YAML File
Start Attendance
Display Excel
Mail Report
Close
Chaitanya Joshi -- SC19M001 -- M.Tech -- Geoinformatics
2019-21

Fig 8. Home Screen of the Face Recognition Attendance System

Creating Training Data -

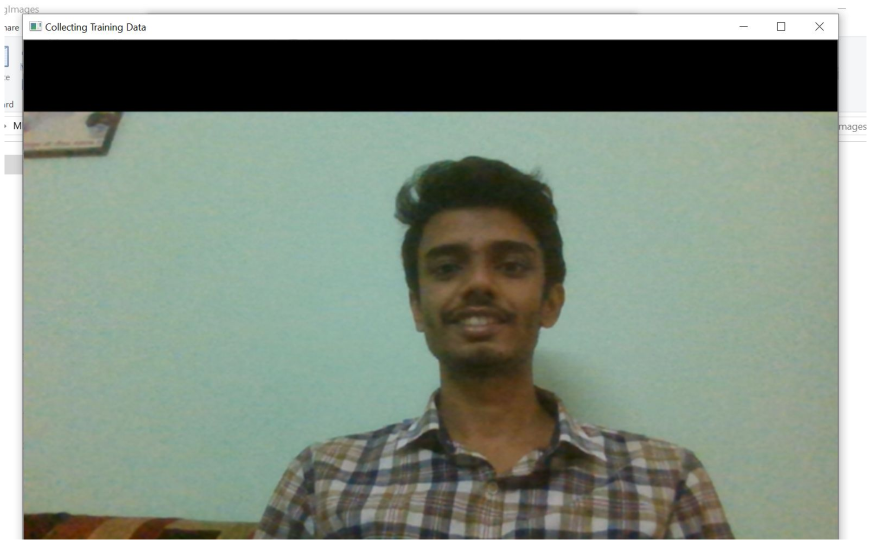
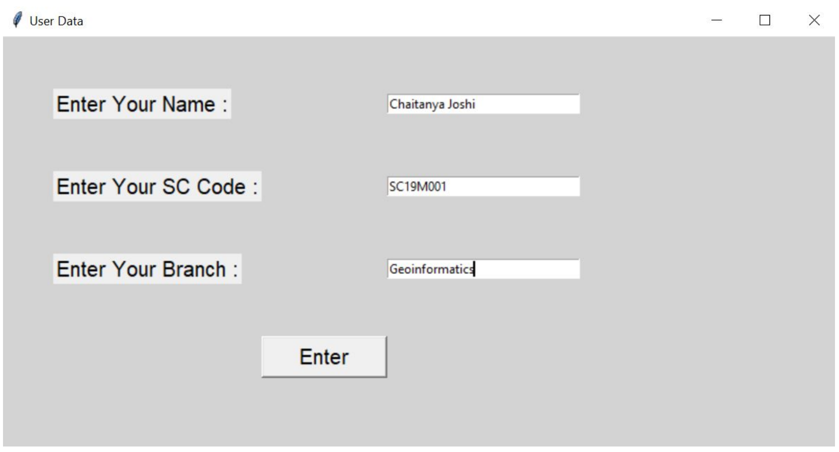
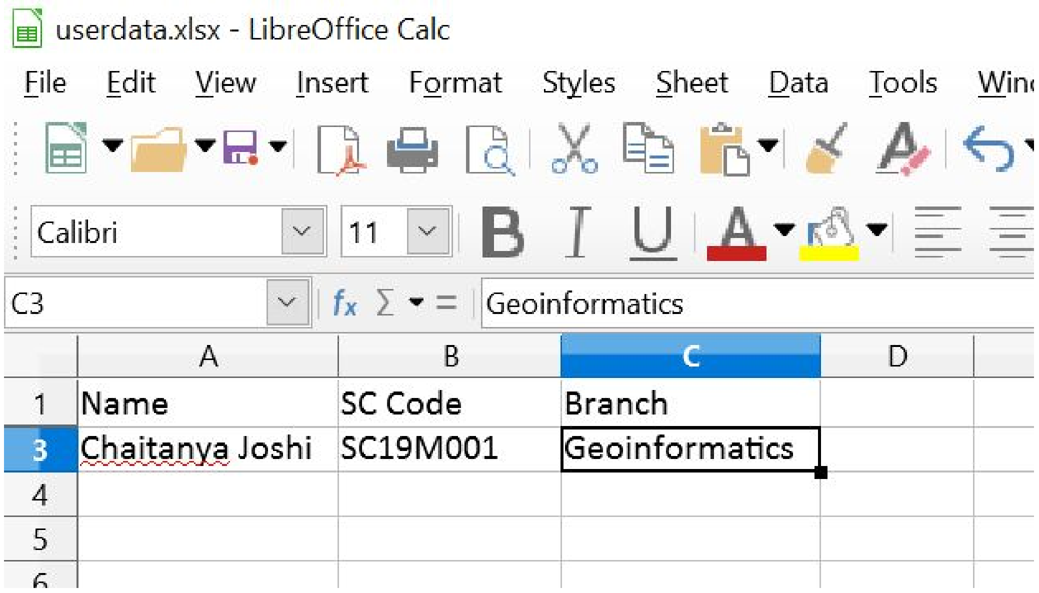


Fig 9. Training Data Creation (200 images are captured)


User Data

Enter Your Name :	Chaitanya Joshi
Enter Your SC Code :	SC19M001
Enter Your Branch :	Geoinformatics

Fig 10. User Data Form


userdata.xlsx - LibreOffice Calc

C3 Geoinformatics

	A	B	C	D
1	Name	SC Code	Branch	
3	Chaitanya Joshi	SC19M001	Geoinformatics	
4				
5				

Fig 11. Userdata.xlsx updated Automatically from User Data Form

Taking Attendance -

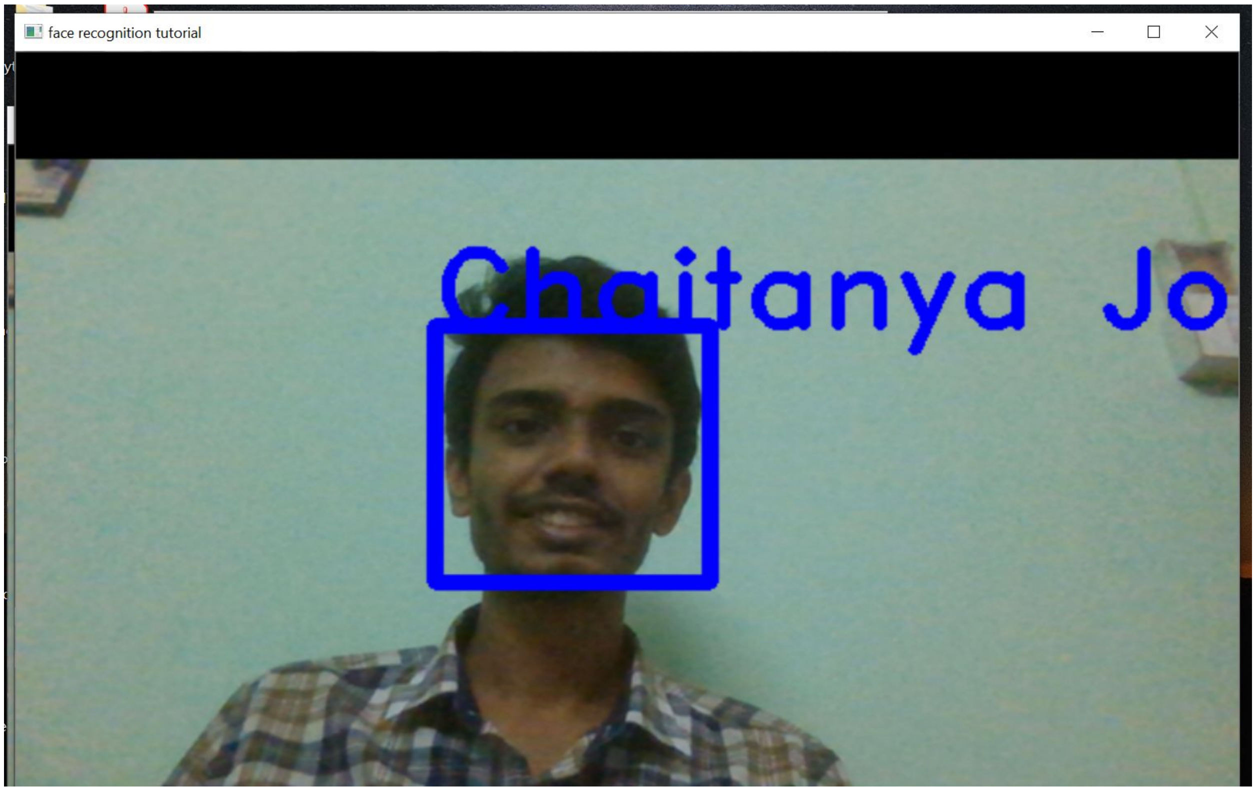
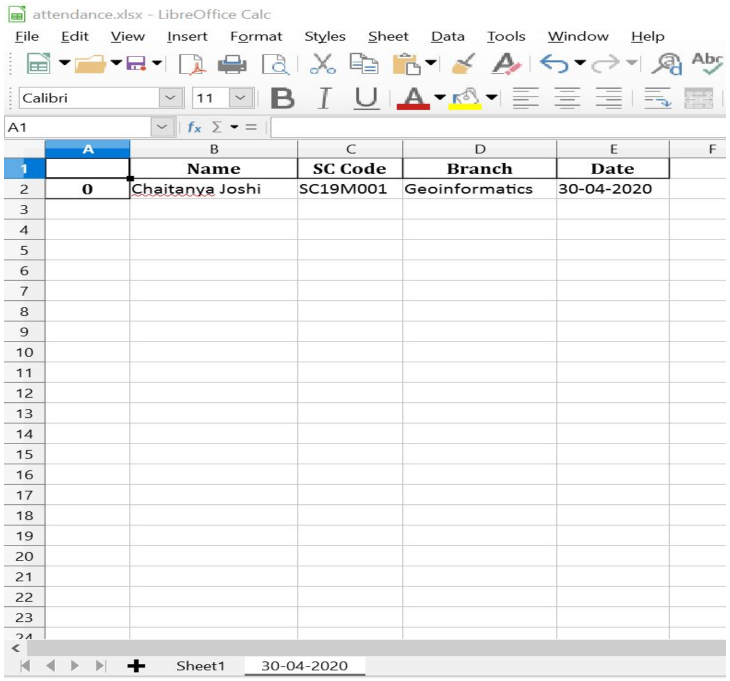


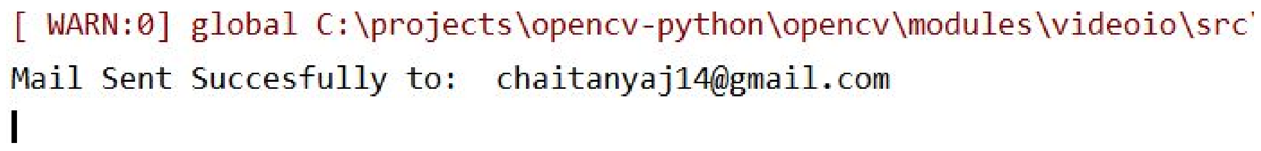
Fig 12. Taking Attendance (recognizing the face) using the Webcam


attendance.xlsx - LibreOffice Calc
File Edit View Insert Format Styles Sheet Data Tools Window Help
Calibri
11
A1
Sheet1
30-04-2020

	Name	SC Code	Branch	Date
0	Chaitanya Joshi	SC19M001	Geoinformatics	30-04-2020

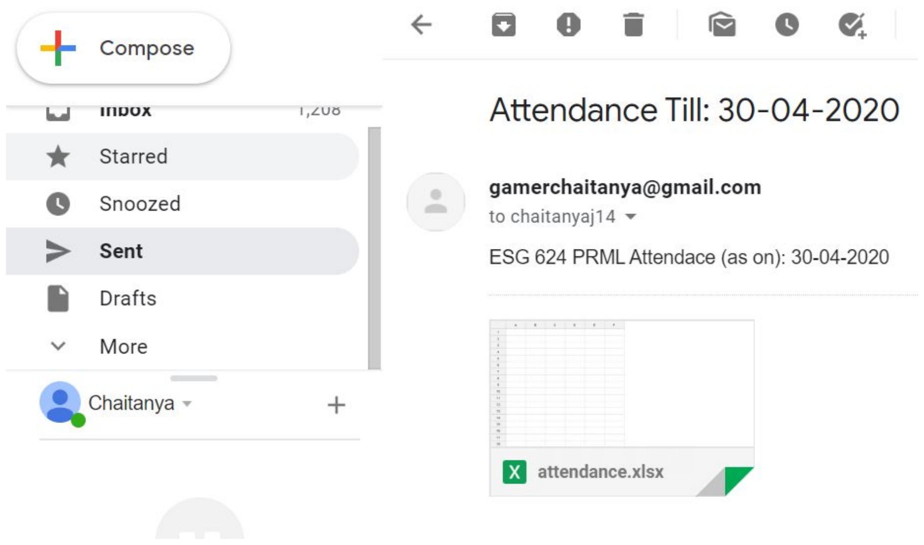
Fig 13. Attendance.xlsx updated having sheet name as the current system date

Send Mail -



```
[ WARN:0] global C:\projects\opencv-python\opencv\modules\videoio\src'
Mail Sent Succesfully to:  chaitanyaj14@gmail.com
```

Fig 14. Mail Sent Confirmation

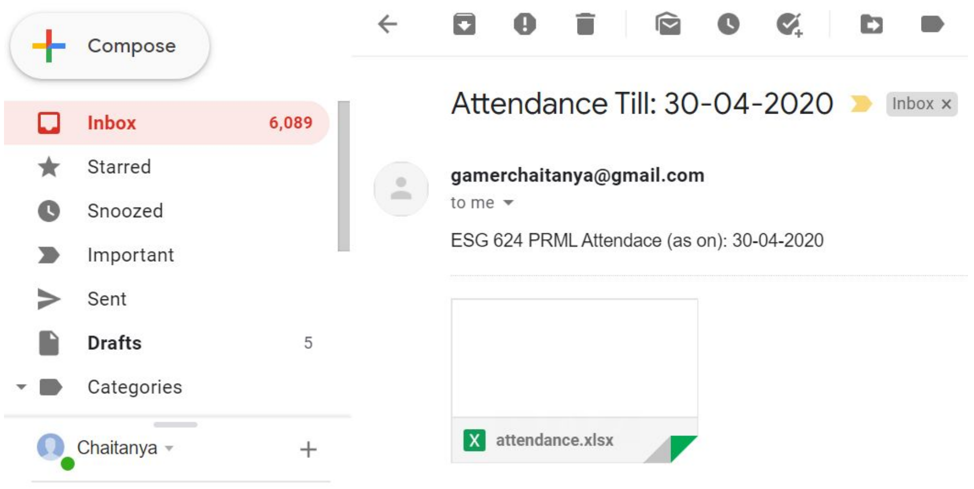


Attendance Till: 30-04-2020

gamerchaitanya@gmail.com
to chaitanyaj14

ESG 624 PRML Attendace (as on): 30-04-2020

Fig 15. Sent Mail



Attendance Till: 30-04-2020

gamerchaitanya@gmail.com
to me

ESG 624 PRML Attendace (as on): 30-04-2020

Fig 16. Received Mail

References -

1. https://courses.cs.washington.edu/courses/cse455/09wi/projects/project4/web/project4.html
2. https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/
3. https://www.youtube.com/watch?v=g4Urfno4aTc
4. https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184
5. https://www.youtube.com/watch?v=h21wMKGs0qs
6. https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/
7. https://www.youtube.com/playlist?list=PLgWKOWHJlDUM_cog-ujJgYoRCJ6LcAhtU
8. https://en.wikipedia.org/wiki/Eigenface
9. https://opencv-python-tutroals.readthedocs.io/en/latest/index.html
10. https://numpy.org/
11. https://www.youtube.com/playlist?list=PLjC8JXsSUrri0XWbCGffJ5to1P40hebu2
12. https://openpyxl.readthedocs.io/en/stable/
13. https://pypi.org/project/xl2dict/
14. https://pandas.pydata.org/
15. https://docs.python.org/3/library/datetime.html
16. https://docs.python.org/3/library/smtplib.html
17. https://docs.python.org/3/library/email.html
18. https://docs.python.org/3/library/tkinter.html