-->go back to index page

ATTENDANCE MONITORING SYSTEM

click here

SUBMITTED BY –

WORK DIVISION (CONTRIBUTION OF TEAMMATES)			
Mukesh	**Vibhav Aakash**	**Sai Koushik**	**VVS Dileep**
18BCE0612	**18BCE2064**	**18BCE0118**	**18BCE0419**
Face recognition	**GUI design**	**Testing, registration and generating csv's**	**Database desgin**
Literature survey	**Literature survey**	**Literature survey**	**Literature survey**

COURSE TITLE : ARTIFICIAL INTELLIGENCE

COURSE CODE : CSE3013

Artificial intelligence Review-3

Under the guidance of

Professor : Harshitha Patel

VIT , Vellore.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

JUNE, 2018

-->go back to index page

INDEX :

-->go back to index page

ABSTRACT :

Attendance Monitoring System is essential in all organizations for checking the performance of students and it is not easy task to check each and every student is present or not. In all organization attendance are taken manually by calling their register numbers or names and noted in attendance registers issued by the department heads as a proof and in some organizations the students wants to sign in these sheets which are stored for future references. This technique is repetitive, complex work and leads to errors as few students regularly sign for their absent students or telling proxy attendance of the absent students. This method additionally makes it more complex to track all the students attendance and difficult to monitoring the individual student attendance in a big classroom atmosphere. In this article, we use are using the technique of utilization face detection and recognition framework to contunisuly recognize students going to class or not and marking their attendance by comparing their faces with database to match and marking attendance

PROBLEM STATEMENT

Develop an attendance management system (face biometry) which helps management to generalise the process and automate things. This should work on with face and also supports manual entry of attendance details.

General problems faced by traditional methods :

1. **Unfair attendance given by students**
2. **Time consumption**
3. **More chances of error occurance.**
4. **Less generalization implies more hard work.**

AIM :

The aim of the project is to develop a software that helps faculty to avoid taking attendance by automatic attendance monitoring system. This Captures the faces of students attending the classes and marks present for them. This simplifies the work consumed by database workers and administration.

This project includes a new approach of detecting faces without using face-recognition module. Every thing is discussed below..

1.Introduction :

Face detection using Haar cascades is a machine learning based approach where a cascade function is trained with a set of input data. OpenCV already contains many pre-trained classifiers for face, eyes, smiles, etc.. Today we will be using the face classifier. You can experiment with other classifiers as well.

You need to download the trained classifier XML file (haarcascade_frontalface_default.xml), which is available in OpenCv's GitHub repository. Save it to your working location.

-->go back to index page

2.LITERATURE SURVEY

- **Akshara Jadhav, Akshay Jadhav Tushar Ladhe, Krishna Yeolekar "Automated Attendance System Using Face Recognition" Irjet Volume: 04 Issue: 01 | Jan -2017**

    1. **The system uses the eigenface approach for face recognition. The method analyzes and computes eigenfaces [4] which are faces composed of eigenvectors.The method also compares the eigenfaces to identify the presence of a person(face) and its identity.The method involves the following steps [1].As a first step the system should be initialized with a set of training faces.Next,when a face is detected the eigenface is calculated for that face.Then, the system compares the eigenvectors of the current face and the stored face image and determines whether the face is identified or not.The final step(optional) is that if the unknown face is detected repeatedly the system may learn to recognize it.**

- **Refik Samet,Muhammed Tanriverdi "Face Recognition-Based Mobile Automatic Classroom Attendance Management System" Published In Ieee 2017 International Conference On Cyberworlds (Cw) (Doi: 10.1109/Cw.2017.34)**

    1. **In the proposed system, RESTful web services were used for communication among teacher, student, and parent applications and the cloud server. Attendance results are stored in a database and accessible by the teacher, student and parent mobile applications.**

- **B. K. Mohamed and C. Raghu - "Fingerprint attendance system for classroom needs", in India Conference (INDICON), 2012 Annual IEEE. IEEE, 2012, pp.433438.**

    1. **There are some problems in conventional attendance tracking system like one is a student missing out their name, while the other leads to a false attendance record. Another issue of having the attendance record in a hardcopy form is that a lecturer may lose the attendance sheet [7]. For student attendance analysis, to obtain the student attendance percentage, manual computation has to be performed by faculty.**

- **S. Konatham, B.S. Chalasani, N. Kulkarni, and T.E. Taeib,"Attendance generating system using RFID and GSM," IEEE Long Island Systems, Applications and Technology Conference (LISAT),2016.**

- **Chaitanya Reddy "Face Recognition Using Artificial Intelligence" Towardsdatascience.Com In April 2017**

    1. **Over the years there were many methods used to implement facial recognition models but thanks to Artificial Intelligence it made our life easier. Using Deep Learning(part of AI), provided with the sufficient data a Facial Recognition System can be built simply and with a high accuracy. We use the a simple Convolutional Neural Networks(CNN) model in order to build a Facial Recognition System.**

- **Facial Recognition System Using Local Binary Patterns(LBP) TS Vishnu Priya, G.Vinitha Sanchez, N.R.Raajan School of Electrical & Electronics Engineering,**

- **Shreyak Sawhney ; Karan Kacker ;Samyak Jain ;Shailendra Narayan Singh ;Rakesh Garg "Real-Time Smart Attendance System Using Face Recognition Techniques" Published In Ieee 2019 9th International Conference On Cloud Computing, Data Science & Engineering (Doi: 10.1109/Confluence.2019.8776934)**

3.OVERVIEW OF THE WORK :

3.1 Objective :

The student's attendance system using artificial intelligence concept mainly works using the concept of facial recognition system. Face is considered as a primary key feature to identify and talk with other peoples in the world because face considered as a unique identity for each and every person. The facial features will be unique to the other individual. Human distinguish a particular persons face based on several factors like color, nose, eyes, ears, etc but for computers, it's difficult to analyze the data so we may use the concept of Computer vision

3.2 SOFTWARE REQUIREMENTS :

1. **Python 3.8.1**
2. **Pip install opencv (CV2)**
3. **pip install numpy**
4. **Tensorflow**
5. **Pillow**
6. **Mysql-connect**
7. **Pandas**

3.3 HARDWARE REQUIREMENTS

1. **Pc with python installed**
2. **Web cam**

4.SYSTEM DESIGN :

4.1 Methodology :

Functional specifications are the requirements in which requires to operate a system. These requirements are necessary to assemble a system which will be required to attain the objectives embarked on previously was implemented by Jireh Robert Jam.Some of the important functional and non functional requirements are outlined below by analyzing the shop keeper story.

- **First capturing the facial image by high qualitity camera o Facial features should be detected in photo. o Crop the overall ranges of faces detected. o Resize all the images until the recognization system takes photo to recognize. o Calculating the overall attendance percentage based on facial features matched. o Storing all the detected face images in a folder. o Loading the images into the database. o We want to train facial features to computer to recognize. o Perform recognition for faces stored on database. o Calculate computer facial recognization speed for effective security. o Performing face recognition sequentially for the each image cropped.**
- **Displaying input and output cropped image side by side on a same slot to recognize and compare the features by machine.**
- **After recognizing the face displays the name of the output image above the image in the given area to identify easily.**

4.2 PROPOSED ALGORITHM:

1. **Capture the Student's Image**
2. **Apply Face detection algorithms to detect face**
3. **Extract the Region Of Interest in Rectangular Bounding Box**
4. **Convert to gray scale, apply histogram equalization and Resize to 100x100 i.e. Apply preprocessing**
5.

```
if Enrollment Phase then
   Store in Database
else
  Apply PCA/LDA/LBPH (For feature Extraction)
Apply Distance Classifier/SVM/Bayesian (for Classification)
end if
```

4.3 FACE DETECTING ALGORITHM:

In order to overcome the problems the algorithm local binary pattern is proposed. Since face image is composed of several minute patterns this can be efficiently identified by applying the local binary pattern operator. The local binary pattern operator is applied on the given face image.

-->go back to index page

METHOD OF LOCAL BINARY PATTERNS(LBP):

In local binary pattern the input face is first converted into the grey image and for that image the binary pattern is calculated by comparing the center pixel with the surrounding pixel.

If the centre pixel is greater than that of the neighboring pixel then it is denoted as 1 and if the neighboring pixel is smaller than that of the centre pixel it is denoted as 0.This should be done for each and every pixel so that we will get the binary pattern.

4.3.1 ADVANTAGES:

- **Improvement of Security Level: Every organization needs to secure their premises for the unknown entry into that place. They also wish to monitor the employees and indusial entry into that place. Those who are entering the organization premises without proper access they are capturing in the security surveillance system and notice to the respective person and alerts instantly concerning the person who doesn't have permission.**

- **Straightforward Integration method :The automatic face detection tools works effectively with the current authentication code that organizations has developed.Bascially the technique is a straightforward to code the system to access organizations automatic data processing which makes the method very clear.**

- **High Accuracy Rates :The main advantage is its Accuracy. The system checks and gives the output without any misunderstanding and bad face detection system. The authorized person will be detected at the right time due to the high accuracy levels. The manual recognition, which is done by securities outside of the organization's premises we may use the face recognition technology to automate the process of identification and assures its perfection without changes. We don't want addition employee to monitor the working of cameras 24/7. The main objective of Automation means to reduce human effect and reducing the cost of employees too. Then any organization can recognizes the fact that usage of automated face identification is highly secure with accurate data.**

5.PROJECT STRUCTURE

- **After run you need to give your face data to system so enter your ID and name in box than click on `Take Images` button.**
- **It will collect 200 images of your faces, it save a images in `TrainingImage` folder**
- **After that we need to train a model(for train a model click on `Train Image` button.**
- **It will take 5-10 minutes for training(for 10 person data).**
- **After training click on `Automatic Attendance` ,it can fill attendance by your face using our trained model (model will save in `TrainingImageLabel`)**
- **it will create `.csv` file of attendance according to time & subject.**

- **You can store data in database (install wampserver),change the DB name according to your in `AMS_Run.py`.**
- **`Manually Fill Attendance` Button in UI is for fill a manually attendance (without face recognition),it's also create a `.csv` and store in a database.**

6.IMPLEMENTATION :

6.1 DESCRIPTION OF MODULES AND PROGRAMS

- **AMS_Run**

Main module where UI exists and every thing about database connectivity and usage of other modules

- **Retrain**

Retraining the model with images that are present in TestingImage

- **Training**

After capturing a set of images every image will be stored in "C:\Users\Mukesh\Documents\AllProjects\python projects\Machine learning" , we can use these images for next step

- **Testing**

Testing of the model is this part

Note : this project uses "haarcascade_frontalface_alt" and "haarcascade_frontalface_default"

6.2 SOURCE CODE :

AMS_RUN.PY

```
import tkinter as tk
from tkinter import *
import cv2
import csv
import os
import numpy as np
from PIL import Image,ImageTk
import pandas as pd
import datetime
```

```
import time

#####Window is our Main frame of system
window = tk.Tk()
window.title("FAMS-Face Recognition Based Attendance Management System")

window.geometry('1280x720')
window.configure(background='snow')

####GUI for manually fill attendance

def manually_fill():
    global sb
    sb = tk.Tk()
    sb.iconbitmap('AMS.ico')
    sb.title("Enter subject name...")
    sb.geometry('580x320')
    sb.configure(background='snow')

    def err_screen_for_subject():

        def ec_delete():
            ec.destroy()
        global ec
        ec = tk.Tk()
        ec.geometry('300x100')
        ec.iconbitmap('AMS.ico')
        ec.title('Warning!!')
        ec.configure(background='snow')
        Label(ec, text='Please enter your subject name!!!', fg='red', bg='white', font=('times', 16, ' bold
')).pack()
        Button(ec, text='OK', command=ec_delete, fg="black", bg="lawn green", width=9, height=1,
activebackground="Red",
               font=('times', 15, ' bold ')).place(x=90, y=50)

    def fill_attendance():
```

```
    ts = time.time()
    Date = datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d')
    timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
    Time = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
    Hour, Minute, Second = timeStamp.split(":")
    ####Creatting csv of attendance

    ##Create table for Attendance
    date_for_DB = datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d')
    global subb
    subb=SUB_ENTRY.get()
    DB_table_name = str(subb + "_" + Date + "_Time_" + Hour + "_" + Minute + "_" + Second)

    import pymysql.connections

    ###Connect to the database
    try:
       global cursor
       connection = pymysql.connect(host='localhost', user='root', password='',
db='manually_fill_attendance')
       cursor = connection.cursor()
    except Exception as e:
       print(e)

    sql = "CREATE TABLE " + DB_table_name + """
               (ID INT NOT NULL AUTO_INCREMENT,
                ENROLLMENT varchar(100) NOT NULL,
                NAME VARCHAR(50) NOT NULL,
                DATE VARCHAR(20) NOT NULL,
                TIME VARCHAR(20) NOT NULL,
                   PRIMARY KEY (ID)
                   );
               """


    try:
```

```
            cursor.execute(sql)  ##for create a table
        except Exception as ex:
            print(ex)  #

        if subb=='':
            err_screen_for_subject()
        else:
            sb.destroy()
            MFW = tk.Tk()
            MFW.iconbitmap('AMS.ico')
            MFW.title("Manually attendance of "+ str(subb))
            MFW.geometry('880x470')
            MFW.configure(background='snow')

            def del_errsc2():
                errsc2.destroy()

            def err_screen1():
                global errsc2
                errsc2 = tk.Tk()
                errsc2.geometry('330x100')
                errsc2.iconbitmap('AMS.ico')
                errsc2.title('Warning!!')
                errsc2.configure(background='snow')
                Label(errsc2, text='Please enter Student & Enrollment!!!', fg='red', bg='white',
                      font=('times', 16, ' bold ')).pack()
                Button(errsc2, text='OK', command=del_errsc2, fg="black", bg="lawn green", width=9,
height=1,
                       activebackground="Red", font=('times', 15, ' bold ')).place(x=90, y=50)

            def testVal(inStr, acttyp):
                if acttyp == '1':  # insert
                    if not inStr.isdigit():
                        return False
                return True
```

```
        ENR = tk.Label(MFW, text="Enter Enrollment", width=15, height=2, fg="white", bg="blue2",
                       font=('times', 15, ' bold '))
        ENR.place(x=30, y=100)

        STU_NAME = tk.Label(MFW, text="Enter Student name", width=15, height=2, fg="white",
bg="blue2",
                            font=('times', 15, ' bold '))
        STU_NAME.place(x=30, y=200)

        global ENR_ENTRY
        ENR_ENTRY = tk.Entry(MFW, width=20,validate='key', bg="yellow", fg="red", font=('times',
23, ' bold '))
        ENR_ENTRY['validatecommand'] = (ENR_ENTRY.register(testVal), '%P', '%d')
        ENR_ENTRY.place(x=290, y=105)

        def remove_enr():
            ENR_ENTRY.delete(first=0, last=22)

        STUDENT_ENTRY = tk.Entry(MFW, width=20, bg="yellow", fg="red", font=('times', 23, ' bold
'))
        STUDENT_ENTRY.place(x=290, y=205)

        def remove_student():
            STUDENT_ENTRY.delete(first=0, last=22)

        ####get important variable
        def enter_data_DB():
            ENROLLMENT = ENR_ENTRY.get()
            STUDENT = STUDENT_ENTRY.get()
            if ENROLLMENT=='':
                err_screen1()
            elif STUDENT=='':
                err_screen1()
            else:
                time = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
                Hour, Minute, Second = time.split(":")
```

```
            Insert_data = "INSERT INTO " + DB_table_name + "
(ID,ENROLLMENT,NAME,DATE,TIME) VALUES (0, %s, %s, %s,%s)"
            VALUES = (str(ENROLLMENT), str(STUDENT), str(Date), str(time))
            try:
               cursor.execute(Insert_data, VALUES)
            except Exception as e:
               print(e)
            ENR_ENTRY.delete(first=0, last=22)
            STUDENT_ENTRY.delete(first=0, last=22)


         def create_csv():
            import csv
            cursor.execute("select * from " + DB_table_name + ";")
            csv_name = 'C:/Users/DELL/Downloads/Compressed/Attendace_management_system-
master/Attendance/Manually Attendance/'+DB_table_name+'.csv'
            with open(csv_name, "w") as csv_file:
               csv_writer = csv.writer(csv_file)
               csv_writer.writerow([i[0] for i in cursor.description])  # write headers
               csv_writer.writerows(cursor)
               O="CSV created Successfully"
               Notifi.configure(text=O, bg="Green", fg="white", width=33, font=('times', 19, 'bold'))
               Notifi.place(x=180, y=380)
            import csv
            import tkinter
            root = tkinter.Tk()
            root.title("Attendance of " + subb)
            root.configure(background='snow')
            with open(csv_name, newline="") as file:
               reader = csv.reader(file)
               r = 0

               for col in reader:
                  c = 0
                  for row in col:
                     # i've added some styling
                     label = tkinter.Label(root, width=13, height=1, fg="black", font=('times', 13, ' bold '),
```

```
                                bg="lawn green", text=row, relief=tkinter.RIDGE)
                label.grid(row=r, column=c)
                c += 1
            r += 1
        root.mainloop()

    Notifi = tk.Label(MFW, text="CSV created Successfully", bg="Green", fg="white", width=33,
                      height=2, font=('times', 19, 'bold'))



    c1ear_enroll = tk.Button(MFW, text="Clear", command=remove_enr, fg="black", bg="deep
pink", width=10,
                             height=1,
                             activebackground="Red", font=('times', 15, ' bold '))
    c1ear_enroll.place(x=690, y=100)

    c1ear_student = tk.Button(MFW, text="Clear", command=remove_student, fg="black",
bg="deep pink", width=10,
                              height=1,
                              activebackground="Red", font=('times', 15, ' bold '))
    c1ear_student.place(x=690, y=200)


    DATA_SUB = tk.Button(MFW, text="Enter Data",command=enter_data_DB, fg="black",
bg="lime green", width=20,
                         height=2,
                         activebackground="Red", font=('times', 15, ' bold '))
    DATA_SUB.place(x=170, y=300)


    MAKE_CSV = tk.Button(MFW, text="Convert to CSV",command=create_csv, fg="black",
bg="red", width=20,
                         height=2,
                         activebackground="Red", font=('times', 15, ' bold '))
    MAKE_CSV.place(x=570, y=300)


    def attf():
        import subprocess
```

```
            subprocess.Popen(r'explorer /select,"C:\Users\kusha\PycharmProjects\Attendace managemnt system\Attendance\Manually Attendance\-------Check atttendance-------"')


        attf = tk.Button(MFW,  text="Check Sheets",command=attf,fg="black"  ,bg="lawn green"  ,width=12  ,height=1 ,activebackground = "Red" ,font=('times', 14, ' bold '))
        attf.place(x=730, y=410)


        MFW.mainloop()



    SUB = tk.Label(sb, text="Enter Subject", width=15, height=2, fg="white", bg="blue2", font=('times', 15, ' bold '))
    SUB.place(x=30, y=100)


    global SUB_ENTRY


    SUB_ENTRY = tk.Entry(sb, width=20, bg="yellow", fg="red", font=('times', 23, ' bold '))
    SUB_ENTRY.place(x=250, y=105)


    fill_manual_attendance = tk.Button(sb, text="Fill Attendance",command=fill_attendance, fg="white", bg="deep pink", width=20, height=2,
                       activebackground="Red", font=('times', 15, ' bold '))
    fill_manual_attendance.place(x=250, y=160)
    sb.mainloop()


##For clear textbox
def clear():
    txt.delete(first=0, last=22)


def clear1():
    txt2.delete(first=0, last=22)
def del_sc1():
    sc1.destroy()
def err_screen():
    global sc1
    sc1 = tk.Tk()
    sc1.geometry('300x100')
```

```
    sc1.iconbitmap('AMS.ico')
    sc1.title('Warning!!')
    sc1.configure(background='snow')
    Label(sc1,text='Enrollment & Name required!!!',fg='red',bg='white',font=('times', 16, ' bold ')).pack()
    Button(sc1,text='OK',command=del_sc1,fg="black"  ,bg="lawn green"  ,width=9  ,height=1,
activebackground = "Red" ,font=('times', 15, ' bold ')).place(x=90,y= 50)


##Error screen2
def del_sc2():
    sc2.destroy()
def err_screen1():
    global sc2
    sc2 = tk.Tk()
    sc2.geometry('300x100')
    sc2.iconbitmap('AMS.ico')
    sc2.title('Warning!!')
    sc2.configure(background='snow')
    Label(sc2,text='Please enter your subject name!!!',fg='red',bg='white',font=('times', 16, ' bold ')).pack()
    Button(sc2,text='OK',command=del_sc2,fg="black"  ,bg="lawn green"  ,width=9  ,height=1,
activebackground = "Red" ,font=('times', 15, ' bold ')).place(x=90,y= 50)


###For take images for datasets
def take_img():
    l1 = txt.get()
    l2 = txt2.get()
    if l1 == '':
        err_screen()
    elif l2 == '':
        err_screen()
    else:
        try:
            cam = cv2.VideoCapture(0)
            detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
            Enrollment = txt.get()
            Name = txt2.get()
            sampleNum = 0
```

```
        while (True):
            ret, img = cam.read()
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            faces = detector.detectMultiScale(gray, 1.3, 5)
            for (x, y, w, h) in faces:
                cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
                # incrementing sample number
                sampleNum = sampleNum + 1
                # saving the captured face in the dataset folder
                cv2.imwrite("TrainingImage/ " + Name + "." + Enrollment + '.' + str(sampleNum) + ".jpg",
                            gray[y:y + h, x:x + w])
                cv2.imshow('Frame', img)
            # wait for 100 miliseconds
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
            # break if the sample number is morethan 100
            elif sampleNum > 70:
                break
        cam.release()
        cv2.destroyAllWindows()
        ts = time.time()
        Date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
        Time = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
        row = [Enrollment, Name, Date, Time]
        with open('StudentDetails\StudentDetails.csv', 'a+') as csvFile:
            writer = csv.writer(csvFile, delimiter=',')
            writer.writerow(row)
            csvFile.close()
        res = "Images Saved for Enrollment : " + Enrollment + " Name : " + Name
        Notification.configure(text=res, bg="SpringGreen3", width=50, font=('times', 18, 'bold'))
        Notification.place(x=250, y=400)
    except FileExistsError as F:
        f = 'Student Data already exists'
        Notification.configure(text=f, bg="Red", width=21)
        Notification.place(x=450, y=400)
```

```
###for choose subject and fill attendance
def subjectchoose():
    def Fillattendances():
        sub=tx.get()
        now = time.time()  ###For calculate seconds of video
        future = now + 20
        if time.time() < future:
            if sub == '':
                err_screen1()
            else:
                recognizer = cv2.face.LBPHFaceRecognizer_create()  # cv2.createLBPHFaceRecognizer()
                try:
                    recognizer.read("TrainingImageLabel\Trainner.yml")
                except:
                    e = 'Model not found,Please train model'
                    Notifica.configure(text=e, bg="red", fg="black", width=33, font=('times', 15, 'bold'))
                    Notifica.place(x=20, y=250)

                harcascadePath = "haarcascade_frontalface_default.xml"
                faceCascade = cv2.CascadeClassifier(harcascadePath)
                df = pd.read_csv("StudentDetails\StudentDetails.csv")
                cam = cv2.VideoCapture(0)
                font = cv2.FONT_HERSHEY_SIMPLEX
                col_names = ['Enrollment', 'Name', 'Date', 'Time']
                attendance = pd.DataFrame(columns=col_names)
                while True:
                    ret, im = cam.read()
                    gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
                    faces = faceCascade.detectMultiScale(gray, 1.2, 5)
                    for (x, y, w, h) in faces:
                        global Id

                        Id, conf = recognizer.predict(gray[y:y + h, x:x + w])
                        if (conf <70):
                            print(conf)
```

```
                global Subject
                global aa
                global date
                global timeStamp
                Subject = tx.get()
                ts = time.time()
                date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
                timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
                aa = df.loc[df['Enrollment'] == Id]['Name'].values
                global tt
                tt = str(Id) + "-" + aa
                En = '15624031' + str(Id)
                attendance.loc[len(attendance)] = [Id, aa, date, timeStamp]
                cv2.rectangle(im, (x, y), (x + w, y + h), (0, 260, 0), 7)
                cv2.putText(im, str(tt), (x + h, y), font, 1, (255, 255, 0,), 4)

            else:
                Id = 'Unknown'
                tt = str(Id)
                cv2.rectangle(im, (x, y), (x + w, y + h), (0, 25, 255), 7)
                cv2.putText(im, str(tt), (x + h, y), font, 1, (0, 25, 255), 4)
        if time.time() > future:
            break

        attendance = attendance.drop_duplicates(['Enrollment'], keep='first')
        cv2.imshow('Filling attedance..', im)
        key = cv2.waitKey(30) & 0xff
        if key == 27:
            break

    ts = time.time()
    date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
    timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
    Hour, Minute, Second = timeStamp.split(":")
    fileName = "Attendance/" + Subject + "_" + date + "_" + Hour + "-" + Minute + "-" + Second
+ ".csv"
```

```
            attendance = attendance.drop_duplicates(['Enrollment'], keep='first')
            print(attendance)
            attendance.to_csv(fileName, index=False)

            ##Create table for Attendance
            date_for_DB = datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d')
            DB_Table_name = str( Subject + "_" + date_for_DB + "_Time_" + Hour + "_" + Minute + "_"
+ Second)
            import pymysql.connections

            ###Connect to the database
            try:
               global cursor
               connection = pymysql.connect(
                  host='localhost', user='root', password='', db='Face_reco_fill')
               cursor = connection.cursor()
            except Exception as e:
               print(e)

            sql = "CREATE TABLE " + DB_Table_name + """
            (ID INT NOT NULL AUTO_INCREMENT,
             ENROLLMENT varchar(100) NOT NULL,
             NAME VARCHAR(50) NOT NULL,
             DATE VARCHAR(20) NOT NULL,
             TIME VARCHAR(20) NOT NULL,
                PRIMARY KEY (ID)
                );
            """
            ####Now enter attendance in Database
            insert_data =  "INSERT INTO " + DB_Table_name +
" (ID,ENROLLMENT,NAME,DATE,TIME) VALUES (0, %s, %s, %s,%s)"
            VALUES = (str(Id), str(aa), str(date), str(timeStamp))
            try:
               cursor.execute(sql)  ##for create a table
               cursor.execute(insert_data, VALUES)##For insert data into table
            except Exception as ex:
```

```
            print(ex)  #


        M = 'Attendance filled Successfully'
        Notifica.configure(text=M, bg="Green", fg="white", width=33, font=('times', 15, 'bold'))
        Notifica.place(x=20, y=250)


        cam.release()
        cv2.destroyAllWindows()


        import csv
        import tkinter
        root = tkinter.Tk()
        root.title("Attendance of " + Subject)
        root.configure(background='snow')
        cs = 'C:/Users/DELL/Downloads/Compressed/Attendace_management_system-master/' +
fileName
        with open(cs, newline="") as file:
            reader = csv.reader(file)
            r = 0


            for col in reader:
                c = 0
                for row in col:
                    # i've added some styling
                    label = tkinter.Label(root, width=8, height=1, fg="black", font=('times', 15, ' bold '),
                                          bg="lawn green", text=row, relief=tkinter.RIDGE)
                    label.grid(row=r, column=c)
                    c += 1
                r += 1
        root.mainloop()
        print(attendance)


    ###windo is frame for subject chooser
    windo = tk.Tk()
    windo.iconbitmap('AMS.ico')
    windo.title("Enter subject name...")
```

```
    windo.geometry('580x320')
    windo.configure(background='snow')
    Notifica = tk.Label(windo, text="Attendance filled Successfully", bg="Green", fg="white", width=33,
                        height=2, font=('times', 15, 'bold'))


    def Attf():
        import subprocess
        subprocess.Popen(r'explorer /select,"C:\Users\kusha\PycharmProjects\Attendace managemnt
system\Attendance\-------Check atttendance-------"')


    attf = tk.Button(windo,  text="Check Sheets",command=Attf,fg="black"  ,bg="lawn
green"  ,width=12  ,height=1 ,activebackground = "Red" ,font=('times', 14, ' bold '))
    attf.place(x=430, y=255)


    sub = tk.Label(windo, text="Enter Subject", width=15, height=2, fg="white", bg="blue2",
font=('times', 15, ' bold '))
    sub.place(x=30, y=100)


    tx = tk.Entry(windo, width=20, bg="yellow", fg="red", font=('times', 23, ' bold '))
    tx.place(x=250, y=105)


    fill_a = tk.Button(windo, text="Fill Attendance", fg="white",command=Fillattendances, bg="deep
pink", width=20, height=2,
                       activebackground="Red", font=('times', 15, ' bold '))
    fill_a.place(x=250, y=160)
    windo.mainloop()


def admin_panel():
    win = tk.Tk()
    win.iconbitmap('AMS.ico')
    win.title("LogIn")
    win.geometry('880x420')
    win.configure(background='snow')


    def log_in():
        username = un_entr.get()
        password = pw_entr.get()
```

```
    if username == 'mukesh':
        if password == 'mukesh':
            win.destroy()
            import csv
            import tkinter
            root = tkinter.Tk()
            root.title("Student Details")
            root.configure(background='snow')

            cs = 'C:/Users/DELL/Downloads/Compressed/Attendace_management_system-
master/StudentDetails/StudentDetails.csv'
            with open(cs, newline="") as file:
                reader = csv.reader(file)
                r = 0

                for col in reader:
                    c = 0
                    for row in col:
                        # i've added some styling
                        label = tkinter.Label(root, width=8, height=1, fg="black", font=('times', 15, ' bold '),
                                              bg="lawn green", text=row, relief=tkinter.RIDGE)
                        label.grid(row=r, column=c)
                        c += 1
                    r += 1
                root.mainloop()
        else:
            valid = 'Incorrect ID or Password'
            Nt.configure(text=valid, bg="red", fg="black", width=38, font=('times', 19, 'bold'))
            Nt.place(x=120, y=350)

    else:
        valid ='Incorrect ID or Password'
        Nt.configure(text=valid, bg="red", fg="black", width=38, font=('times', 19, 'bold'))
        Nt.place(x=120, y=350)
```

```
Nt = tk.Label(win, text="Attendance filled Successfully", bg="Green", fg="white", width=40,
              height=2, font=('times', 19, 'bold'))
# Nt.place(x=120, y=350)

un = tk.Label(win, text="Enter username", width=15, height=2, fg="white", bg="blue2",
               font=('times', 15, ' bold '))
un.place(x=30, y=50)

pw = tk.Label(win, text="Enter password", width=15, height=2, fg="white", bg="blue2",
              font=('times', 15, ' bold '))
pw.place(x=30, y=150)

def c00():
   un_entr.delete(first=0, last=22)

un_entr = tk.Entry(win, width=20, bg="yellow", fg="red", font=('times', 23, ' bold '))
un_entr.place(x=290, y=55)

def c11():
   pw_entr.delete(first=0, last=22)

pw_entr = tk.Entry(win, width=20,show="*", bg="yellow", fg="red", font=('times', 23, ' bold '))
pw_entr.place(x=290, y=155)

c0 = tk.Button(win, text="Clear", command=c00, fg="black", bg="deep pink", width=10, height=1,
                    activebackground="Red", font=('times', 15, ' bold '))
c0.place(x=690, y=55)

c1 = tk.Button(win, text="Clear", command=c11, fg="black", bg="deep pink", width=10, height=1,
               activebackground="Red", font=('times', 15, ' bold '))
c1.place(x=690, y=155)

Login = tk.Button(win, text="LogIn", fg="black", bg="lime green", width=20,
                  height=2,
                  activebackground="Red",command=log_in, font=('times', 15, ' bold '))
```

```
    Login.place(x=290, y=250)
    win.mainloop()



###For train the model
def trainimg():
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    global detector
    detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
    try:
        global faces,Id
        faces, Id = getImagesAndLabels("TrainingImage")
    except Exception as e:
        l='please make "TrainingImage" folder & put Images'
        Notification.configure(text=l, bg="SpringGreen3", width=50, font=('times', 18, 'bold'))
        Notification.place(x=350, y=400)

    recognizer.train(faces, np.array(Id))
    try:
        recognizer.save("TrainingImageLabel\Trainner.yml")
    except Exception as e:
        q='Please make "TrainingImageLabel" folder'
        Notification.configure(text=q, bg="SpringGreen3", width=50, font=('times', 18, 'bold'))
        Notification.place(x=350, y=400)

    res = "Model Trained"  # +",".join(str(f) for f in Id)
    Notification.configure(text=res, bg="SpringGreen3", width=50, font=('times', 18, 'bold'))
    Notification.place(x=250, y=400)

def getImagesAndLabels(path):
    imagePaths = [os.path.join(path, f) for f in os.listdir(path)]
    # create empth face list
    faceSamples = []
    # create empty ID list
    Ids = []
    # now looping through all the image paths and loading the Ids and the images
```

```
    for imagePath in imagePaths:
        # loading the image and converting it to gray scale
        pilImage = Image.open(imagePath).convert('L')
        # Now we are converting the PIL image into numpy array
        imageNp = np.array(pilImage, 'uint8')
        # getting the Id from the image

        Id = int(os.path.split(imagePath)[-1].split(".")[1])
        # extract the face from the training image sample
        faces = detector.detectMultiScale(imageNp)
        # If a face is there then append that in the list as well as Id of it
        for (x, y, w, h) in faces:
            faceSamples.append(imageNp[y:y + h, x:x + w])
            Ids.append(Id)
    return faceSamples, Ids


window.grid_rowconfigure(0, weight=1)
window.grid_columnconfigure(0, weight=1)
window.iconbitmap('AMS.ico')


def on_closing():
    from tkinter import messagebox
    if messagebox.askokcancel("Quit", "Do you want to quit?"):
        window.destroy()
window.protocol("WM_DELETE_WINDOW", on_closing)


message = tk.Label(window, text="Face-Recognition-Based-Attendance-Management-System",
bg="cyan", fg="black", width=50,
                   height=3, font=('times', 30, 'italic bold '))


message.place(x=80, y=20)


Notification = tk.Label(window, text="All things good", bg="Green", fg="white", width=15,
                        height=3, font=('times', 17, 'bold'))
```

```
lbl = tk.Label(window, text="Enter Enrollment", width=20, height=2, fg="black", bg="deep pink", font=('times', 15, ' bold '))
lbl.place(x=200, y=200)

def testVal(inStr,acttyp):
    if acttyp == '1': #insert
        if not inStr.isdigit():
            return False
    return True

txt = tk.Entry(window, validate="key", width=20, bg="yellow", fg="red", font=('times', 25, ' bold '))
txt['validatecommand'] = (txt.register(testVal),'%P','%d')
txt.place(x=550, y=210)

lbl2 = tk.Label(window, text="Enter Name", width=20, fg="black", bg="deep pink", height=2, font=('times', 15, ' bold '))
lbl2.place(x=200, y=300)

txt2 = tk.Entry(window, width=20, bg="yellow", fg="red", font=('times', 25, ' bold '))
txt2.place(x=550, y=310)

clearButton = tk.Button(window, text="Clear",command=clear,fg="black"  ,bg="deep pink"  ,width=10  ,height=1 ,activebackground = "Red" ,font=('times', 15, ' bold '))
clearButton.place(x=950, y=210)

clearButton1 = tk.Button(window, text="Clear",command=clear1,fg="black"  ,bg="deep pink"  ,width=10 ,height=1, activebackground = "Red" ,font=('times', 15, ' bold '))
clearButton1.place(x=950, y=310)

AP = tk.Button(window, text="Check Register students",command=admin_panel,fg="black"  ,bg="cyan"  ,width=19 ,height=1, activebackground = "Red" ,font=('times', 15, ' bold '))
AP.place(x=990, y=410)

takeImg = tk.Button(window, text="Take Images",command=take_img,fg="white"  ,bg="blue2"  ,width=20  ,height=3, activebackground = "Red" ,font=('times', 15, ' bold '))
takeImg.place(x=90, y=500)
```

-->go back to index page

```
trainImg = tk.Button(window, text="Train Images",fg="black",command=trainimg ,bg="lawn green"  ,width=20  ,height=3, activebackground = "Red" ,font=('times', 15, ' bold '))
trainImg.place(x=390, y=500)


FA = tk.Button(window, text="Automatic Attendace",fg="white",command=subjectchoose  ,bg="blue2"  ,width=20  ,height=3, activebackground = "Red" ,font=('times', 15, ' bold '))
FA.place(x=690, y=500)


quitWindow = tk.Button(window, text="Manually Fill Attendance", command=manually_fill  ,fg="black"  ,bg="lawn green"  ,width=20  ,height=3, activebackground = "Red" ,font=('times', 15, ' bold '))
quitWindow.place(x=990, y=500)


window.mainloop()
```

RETRAIN.PY

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import collections
from datetime import datetime
import hashlib
import os.path
import random
import re
import sys
import tarfile

import numpy as np
from six.moves import urllib
import tensorflow as tf
```

```
from tensorflow.python.framework import graph_util
from tensorflow.python.framework import tensor_shape
from tensorflow.python.platform import gfile
from tensorflow.python.util import compat

FLAGS = None
MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1  # ~134M
def create_image_lists(image_dir, testing_percentage, validation_percentage):
 if not gfile.Exists(image_dir):
  tf.logging.error("Image directory '" + image_dir + "' not found.")
  return None
 result = collections.OrderedDict()
 sub_dirs = [
  os.path.join(image_dir,item)
  for item in gfile.ListDirectory(image_dir)]
 sub_dirs = sorted(item for item in sub_dirs
                if gfile.IsDirectory(item))
 for sub_dir in sub_dirs:
  extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']
  file_list = []
  dir_name = os.path.basename(sub_dir)
  if dir_name == image_dir:
   continue
  tf.logging.info("Looking for images in '" + dir_name + "'")
  for extension in extensions:
   file_glob = os.path.join(image_dir, dir_name, '*.' + extension)
   file_list.extend(gfile.Glob(file_glob))
  if not file_list:
   tf.logging.warning('No files found')
   continue
  if len(file_list) < 20:
   tf.logging.warning(
      'WARNING: Folder has less than 20 images, which may cause issues.')
  elif len(file_list) > MAX_NUM_IMAGES_PER_CLASS:
   tf.logging.warning(
      'WARNING: Folder {} has more than {} images. Some images will '
```

```
        'never be selected.'.format(dir_name, MAX_NUM_IMAGES_PER_CLASS))
    label_name = re.sub(r'[^a-z0-9]+', ' ', dir_name.lower())
    training_images = []
    testing_images = []
    validation_images = []
    for file_name in file_list:
      base_name = os.path.basename(file_name)
      hash_name = re.sub(r'_nohash_.*$', '', file_name)
      hash_name_hashed = hashlib.sha1(compat.as_bytes(hash_name)).hexdigest()
      percentage_hash = ((int(hash_name_hashed, 16) %
                          (MAX_NUM_IMAGES_PER_CLASS + 1)) *
                         (100.0 / MAX_NUM_IMAGES_PER_CLASS))
      if percentage_hash < validation_percentage:
        validation_images.append(base_name)
      elif percentage_hash < (testing_percentage + validation_percentage):
        testing_images.append(base_name)
      else:
        training_images.append(base_name)
    result[label_name] = {
        'dir': dir_name,
        'training': training_images,
        'testing': testing_images,
        'validation': validation_images,
    }
  return result


def get_image_path(image_lists, label_name, index, image_dir, category):
  if label_name not in image_lists:
    tf.logging.fatal('Label does not exist %s.', label_name)
  label_lists = image_lists[label_name]
  if category not in label_lists:
    tf.logging.fatal('Category does not exist %s.', category)
  category_list = label_lists[category]
  if not category_list:
    tf.logging.fatal('Label %s has no images in the category %s.',
```

```
                label_name, category)
  mod_index = index % len(category_list)
  base_name = category_list[mod_index]
  sub_dir = label_lists['dir']
  full_path = os.path.join(image_dir, sub_dir, base_name)
  return full_path


def get_bottleneck_path(image_lists, label_name, index, bottleneck_dir,
                        category, architecture):
  """Returns a path to a bottleneck file for a label at the given index.

  Args:
    image_lists: Dictionary of training images for each label.
    label_name: Label string we want to get an image for.
    index: Integer offset of the image we want. This will be moduloed by the
    available number of images for the label, so it can be arbitrarily large.
    bottleneck_dir: Folder string holding cached files of bottleneck values.
    category: Name string of set to pull images from - training, testing, or
    validation.
    architecture: The name of the model architecture.

  Returns:
    File system path string to an image that meets the requested parameters.
  """
  return get_image_path(image_lists, label_name, index, bottleneck_dir,
                        category) + '_' + architecture + '.txt'


def create_model_graph(model_info):
  """Creates a graph from saved GraphDef file and returns a Graph object.

  Args:
    model_info: Dictionary containing information about the model architecture.

  Returns:
```

```
  Graph holding the trained Inception network, and various tensors we'll be
  manipulating.
  """
  with tf.Graph().as_default() as graph:
    model_path = os.path.join(FLAGS.model_dir, model_info['model_file_name'])
    with gfile.FastGFile(model_path, 'rb') as f:
      graph_def = tf.GraphDef()
      graph_def.ParseFromString(f.read())
      bottleneck_tensor, resized_input_tensor = (tf.import_graph_def(
          graph_def,
          name='',
          return_elements=[
              model_info['bottleneck_tensor_name'],
              model_info['resized_input_tensor_name'],
          ]))
  return graph, bottleneck_tensor, resized_input_tensor


def run_bottleneck_on_image(sess, image_data, image_data_tensor,
                            decoded_image_tensor, resized_input_tensor,
                            bottleneck_tensor):
  """Runs inference on an image to extract the 'bottleneck' summary layer.

  Args:
    sess: Current active TensorFlow Session.
    image_data: String of raw JPEG data.
    image_data_tensor: Input data layer in the graph.
    decoded_image_tensor: Output of initial image resizing and  preprocessing.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: Layer before the final softmax.

  Returns:
    Numpy array of bottleneck values.
  """
  # First decode the JPEG image, resize it, and rescale the pixel values.
  resized_input_values = sess.run(decoded_image_tensor,
```

```
                        {image_data_tensor: image_data})
  # Then run it through the recognition network.
  bottleneck_values = sess.run(bottleneck_tensor,
                               {resized_input_tensor: resized_input_values})
  bottleneck_values = np.squeeze(bottleneck_values)
  return bottleneck_values


def maybe_download_and_extract(data_url):
  """Download and extract model tar file.

  If the pretrained model we're using doesn't already exist, this function
  downloads it from the TensorFlow.org website and unpacks it into a directory.

  Args:
    data_url: Web location of the tar file containing the pretrained model.
  """
  dest_directory = FLAGS.model_dir
  if not os.path.exists(dest_directory):
    os.makedirs(dest_directory)
  filename = data_url.split('/')[-1]
  filepath = os.path.join(dest_directory, filename)
  if not os.path.exists(filepath):

    def _progress(count, block_size, total_size):
      sys.stdout.write('\r>> Downloading %s %.1f%%' %
                       (filename,
                        float(count * block_size) / float(total_size) * 100.0))
      sys.stdout.flush()

    filepath, _ = urllib.request.urlretrieve(data_url, filepath, _progress)
    print()
    statinfo = os.stat(filepath)
    tf.logging.info('Successfully downloaded', filename, statinfo.st_size,
                    'bytes.')
  tarfile.open(filepath, 'r:gz').extractall(dest_directory)
```

```
def ensure_dir_exists(dir_name):
  """Makes sure the folder exists on disk.

  Args:
    dir_name: Path string to the folder we want to create.
  """
  if not os.path.exists(dir_name):
    os.makedirs(dir_name)


bottleneck_path_2_bottleneck_values = {}


def create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           decoded_image_tensor, resized_input_tensor,
                           bottleneck_tensor):
  """Create a single bottleneck file."""
  tf.logging.info('Creating bottleneck at ' + bottleneck_path)
  image_path = get_image_path(image_lists, label_name, index,
                              image_dir, category)
  if not gfile.Exists(image_path):
    tf.logging.fatal('File does not exist %s', image_path)
  image_data = gfile.FastGFile(image_path, 'rb').read()
  try:
    bottleneck_values = run_bottleneck_on_image(
        sess, image_data, jpeg_data_tensor, decoded_image_tensor,
        resized_input_tensor, bottleneck_tensor)
  except Exception as e:
    raise RuntimeError('Error during processing file %s (%s)' % (image_path,
                                                                 str(e)))
  bottleneck_string = ','.join(str(x) for x in bottleneck_values)
  with open(bottleneck_path, 'w') as bottleneck_file:
    bottleneck_file.write(bottleneck_string)
```

```
def get_or_create_bottleneck(sess, image_lists, label_name, index, image_dir,
                             category, bottleneck_dir, jpeg_data_tensor,
                             decoded_image_tensor, resized_input_tensor,
                             bottleneck_tensor, architecture):
  """Retrieves or calculates bottleneck values for an image.

  If a cached version of the bottleneck data exists on-disk, return that,
  otherwise calculate the data and save it to disk for future use.

  Args:
    sess: The current active TensorFlow Session.
    image_lists: Dictionary of training images for each label.
    label_name: Label string we want to get an image for.
    index: Integer offset of the image we want. This will be modulo-ed by the
    available number of images for the label, so it can be arbitrarily large.
    image_dir: Root folder string  of the subfolders containing the training
    images.
    category: Name string of which  set to pull images from - training, testing,
    or validation.
    bottleneck_dir: Folder string holding cached files of bottleneck values.
    jpeg_data_tensor: The tensor to feed loaded jpeg data into.
    decoded_image_tensor: The output of decoding and resizing the image.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The output tensor for the bottleneck values.
    architecture: The name of the model architecture.

  Returns:
    Numpy array of values produced by the bottleneck layer for the image.
  """
  label_lists = image_lists[label_name]
  sub_dir = label_lists['dir']
  sub_dir_path = os.path.join(bottleneck_dir, sub_dir)
  ensure_dir_exists(sub_dir_path)
  bottleneck_path = get_bottleneck_path(image_lists, label_name, index,
```

```
                                bottleneck_dir, category, architecture)
  if not os.path.exists(bottleneck_path):
    create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           decoded_image_tensor, resized_input_tensor,
                           bottleneck_tensor)
  with open(bottleneck_path, 'r') as bottleneck_file:
    bottleneck_string = bottleneck_file.read()
  did_hit_error = False
  try:
    bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
  except ValueError:
    tf.logging.warning('Invalid float found, recreating bottleneck')
    did_hit_error = True
  if did_hit_error:
    create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           decoded_image_tensor, resized_input_tensor,
                           bottleneck_tensor)
    with open(bottleneck_path, 'r') as bottleneck_file:
      bottleneck_string = bottleneck_file.read()
    # Allow exceptions to propagate here, since they shouldn't happen after a
    # fresh creation
    bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
  return bottleneck_values


def cache_bottlenecks(sess, image_lists, image_dir, bottleneck_dir,
                      jpeg_data_tensor, decoded_image_tensor,
                      resized_input_tensor, bottleneck_tensor, architecture):
  """Ensures all the training, testing, and validation bottlenecks are cached.

  Because we're likely to read the same image multiple times (if there are no
  distortions applied during training) it can speed things up a lot if we
  calculate the bottleneck layer values once for each image during
  preprocessing, and then just read those cached values repeatedly during
```

```
  training. Here we go through all the images we've found, calculate those
  values, and save them off.

  Args:
    sess: The current active TensorFlow Session.
    image_lists: Dictionary of training images for each label.
    image_dir: Root folder string of the subfolders containing the training
    images.
    bottleneck_dir: Folder string holding cached files of bottleneck values.
    jpeg_data_tensor: Input tensor for jpeg data from file.
    decoded_image_tensor: The output of decoding and resizing the image.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The penultimate output layer of the graph.
    architecture: The name of the model architecture.

  Returns:
    Nothing.
  """
  how_many_bottlenecks = 0
  ensure_dir_exists(bottleneck_dir)
  for label_name, label_lists in image_lists.items():
    for category in ['training', 'testing', 'validation']:
      category_list = label_lists[category]
      for index, unused_base_name in enumerate(category_list):
        get_or_create_bottleneck(
            sess, image_lists, label_name, index, image_dir, category,
            bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
            resized_input_tensor, bottleneck_tensor, architecture)

        how_many_bottlenecks += 1
        if how_many_bottlenecks % 100 == 0:
          tf.logging.info(
              str(how_many_bottlenecks) + ' bottleneck files created.')


def get_random_cached_bottlenecks(sess, image_lists, how_many, category,
```

```
                    bottleneck_dir, image_dir, jpeg_data_tensor,
                    decoded_image_tensor, resized_input_tensor,
                    bottleneck_tensor, architecture):
  """Retrieves bottleneck values for cached images.

  If no distortions are being applied, this function can retrieve the cached
  bottleneck values directly from disk for images. It picks a random set of
  images from the specified category.

  Args:
    sess: Current TensorFlow Session.
    image_lists: Dictionary of training images for each label.
    how_many: If positive, a random sample of this size will be chosen.
    If negative, all bottlenecks will be retrieved.
    category: Name string of which set to pull from - training, testing, or
    validation.
    bottleneck_dir: Folder string holding cached files of bottleneck values.
    image_dir: Root folder string of the subfolders containing the training
    images.
    jpeg_data_tensor: The layer to feed jpeg image data into.
    decoded_image_tensor: The output of decoding and resizing the image.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The bottleneck output layer of the CNN graph.
    architecture: The name of the model architecture.

  Returns:
    List of bottleneck arrays, their corresponding ground truths, and the
    relevant filenames.
  """
  class_count = len(image_lists.keys())
  bottlenecks = []
  ground_truths = []
  filenames = []
  if how_many >= 0:
    # Retrieve a random sample of bottlenecks.
    for unused_i in range(how_many):
```

```
    label_index = random.randrange(class_count)
    label_name = list(image_lists.keys())[label_index]
    image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
    image_name = get_image_path(image_lists, label_name, image_index,
                                image_dir, category)
    bottleneck = get_or_create_bottleneck(
        sess, image_lists, label_name, image_index, image_dir, category,
        bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
        resized_input_tensor, bottleneck_tensor, architecture)
    ground_truth = np.zeros(class_count, dtype=np.float32)
    ground_truth[label_index] = 1.0
    bottlenecks.append(bottleneck)
    ground_truths.append(ground_truth)
    filenames.append(image_name)
  else:
    # Retrieve all bottlenecks.
    for label_index, label_name in enumerate(image_lists.keys()):
      for image_index, image_name in enumerate(
          image_lists[label_name][category]):
        image_name = get_image_path(image_lists, label_name, image_index,
                                    image_dir, category)
        bottleneck = get_or_create_bottleneck(
            sess, image_lists, label_name, image_index, image_dir, category,
            bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
            resized_input_tensor, bottleneck_tensor, architecture)
        ground_truth = np.zeros(class_count, dtype=np.float32)
        ground_truth[label_index] = 1.0
        bottlenecks.append(bottleneck)
        ground_truths.append(ground_truth)
        filenames.append(image_name)
  return bottlenecks, ground_truths, filenames


def get_random_distorted_bottlenecks(
    sess, image_lists, how_many, category, image_dir, input_jpeg_tensor,
    distorted_image, resized_input_tensor, bottleneck_tensor):
```

```
"""Retrieves bottleneck values for training images, after distortions.

If we're training with distortions like crops, scales, or flips, we have to
recalculate the full model for every image, and so we can't use cached
bottleneck values. Instead we find random images for the requested category,
run them through the distortion graph, and then the full graph to get the
bottleneck results for each.

Args:
  sess: Current TensorFlow Session.
  image_lists: Dictionary of training images for each label.
  how_many: The integer number of bottleneck values to return.
  category: Name string of which set of images to fetch - training, testing,
  or validation.
  image_dir: Root folder string of the subfolders containing the training
  images.
  input_jpeg_tensor: The input layer we feed the image data to.
  distorted_image: The output node of the distortion graph.
  resized_input_tensor: The input node of the recognition graph.
  bottleneck_tensor: The bottleneck output layer of the CNN graph.

Returns:
  List of bottleneck arrays and their corresponding ground truths.
"""
class_count = len(image_lists.keys())
bottlenecks = []
ground_truths = []
for unused_i in range(how_many):
  label_index = random.randrange(class_count)
  label_name = list(image_lists.keys())[label_index]
  image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
  image_path = get_image_path(image_lists, label_name, image_index, image_dir,
                              category)
  if not gfile.Exists(image_path):
    tf.logging.fatal('File does not exist %s', image_path)
  jpeg_data = gfile.FastGFile(image_path, 'rb').read()
```

```
  # Note that we materialize the distorted_image_data as a numpy array before
  # sending running inference on the image. This involves 2 memory copies and
  # might be optimized in other implementations.
  distorted_image_data = sess.run(distorted_image,
                                  {input_jpeg_tensor: jpeg_data})
  bottleneck_values = sess.run(bottleneck_tensor,
                               {resized_input_tensor: distorted_image_data})
  bottleneck_values = np.squeeze(bottleneck_values)
  ground_truth = np.zeros(class_count, dtype=np.float32)
  ground_truth[label_index] = 1.0
  bottlenecks.append(bottleneck_values)
  ground_truths.append(ground_truth)
return bottlenecks, ground_truths


def should_distort_images(flip_left_right, random_crop, random_scale,
                          random_brightness):
 """Whether any distortions are enabled, from the input flags.

 Args:
  flip_left_right: Boolean whether to randomly mirror images horizontally.
  random_crop: Integer percentage setting the total margin used around the
  crop box.
  random_scale: Integer percentage of how much to vary the scale by.
  random_brightness: Integer range to randomly multiply the pixel values by.

 Returns:
  Boolean value indicating whether any distortions should be applied.
 """
 return (flip_left_right or (random_crop != 0) or (random_scale != 0) or
         (random_brightness != 0))


def add_input_distortions(flip_left_right, random_crop, random_scale,
                          random_brightness, input_width, input_height,
                          input_depth, input_mean, input_std):
```

"""Creates the operations to apply the specified distortions.

During training it can help to improve the results if we run the images through simple distortions like crops, scales, and flips. These reflect the kind of variations we expect in the real world, and so can help train the model to cope with natural data more effectively. Here we take the supplied parameters and construct a network of operations to apply them to an image.

Cropping

~~~~~~~~

Cropping is done by placing a bounding box at a random position in the full image. The cropping parameter controls the size of that box relative to the input image. If it's zero, then the box is the same size as the input and no cropping is performed. If the value is 50%, then the crop box will be half the width and height of the input. In a diagram it looks like this:

```
<      width        >
+---------------------+
|                     |
|   width - crop%     |
|    <     >          |
|    +------+         |
|    |      |         |
|    |      |         |
|    |      |         |
|    +------+         |
|                     |
|                     |
+---------------------+
```

Scaling

~~~~~~~

Scaling is a lot like cropping, except that the bounding box is always centered and its size varies randomly within the given range. For example if

```
    the scale percentage is zero, then the bounding box is the same size as the
    input and no scaling is applied. If it's 50%, then the bounding box will be in
    a random range between half the width and height and full size.

    Args:
      flip_left_right: Boolean whether to randomly mirror images horizontally.
      random_crop: Integer percentage setting the total margin used around the
      crop box.
      random_scale: Integer percentage of how much to vary the scale by.
      random_brightness: Integer range to randomly multiply the pixel values by.
      graph.
      input_width: Horizontal size of expected input image to model.
      input_height: Vertical size of expected input image to model.
      input_depth: How many channels the expected input image should have.
      input_mean: Pixel value that should be zero in the image for the graph.
      input_std: How much to divide the pixel values by before recognition.

    Returns:
      The jpeg input layer and the distorted result tensor.
    """

    jpeg_data = tf.placeholder(tf.string, name='DistortJPGInput')
    decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_depth)
    decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
    decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
    margin_scale = 1.0 + (random_crop / 100.0)
    resize_scale = 1.0 + (random_scale / 100.0)
    margin_scale_value = tf.constant(margin_scale)
    resize_scale_value = tf.random_uniform(tensor_shape.scalar(),
                                           minval=1.0,
                                           maxval=resize_scale)
    scale_value = tf.multiply(margin_scale_value, resize_scale_value)
    precrop_width = tf.multiply(scale_value, input_width)
    precrop_height = tf.multiply(scale_value, input_height)
    precrop_shape = tf.stack([precrop_height, precrop_width])
    precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32)
```

```
  precropped_image = tf.image.resize_bilinear(decoded_image_4d,
                                              precrop_shape_as_int)
  precropped_image_3d = tf.squeeze(precropped_image, squeeze_dims=[0])
  cropped_image = tf.random_crop(precropped_image_3d,
                                 [input_height, input_width, input_depth])
  if flip_left_right:
    flipped_image = tf.image.random_flip_left_right(cropped_image)
  else:
    flipped_image = cropped_image
  brightness_min = 1.0 - (random_brightness / 100.0)
  brightness_max = 1.0 + (random_brightness / 100.0)
  brightness_value = tf.random_uniform(tensor_shape.scalar(),
                                       minval=brightness_min,
                                       maxval=brightness_max)
  brightened_image = tf.multiply(flipped_image, brightness_value)
  offset_image = tf.subtract(brightened_image, input_mean)
  mul_image = tf.multiply(offset_image, 1.0 / input_std)
  distort_result = tf.expand_dims(mul_image, 0, name='DistortResult')
  return jpeg_data, distort_result


def variable_summaries(var):
  """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean', mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    tf.summary.scalar('stddev', stddev)
    tf.summary.scalar('max', tf.reduce_max(var))
    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)


def add_final_training_ops(class_count, final_tensor_name, bottleneck_tensor,
                           bottleneck_tensor_size):
```

```
"""Adds a new softmax and fully-connected layer for training.

We need to retrain the top layer to identify our new classes, so this function
adds the right operations to the graph, along with some variables to hold the
weights, and then sets up all the gradients for the backward pass.

The set up for the softmax and fully-connected layers is based on:
https://www.tensorflow.org/versions/master/tutorials/mnist/beginners/index.html

Args:
  class_count: Integer of how many categories of things we're trying to
  recognize.
  final_tensor_name: Name string for the new final node that produces results.
  bottleneck_tensor: The output of the main CNN graph.
  bottleneck_tensor_size: How many entries in the bottleneck vector.

Returns:
  The tensors for the training and cross entropy results, and tensors for the
  bottleneck input and ground truth input.
"""
with tf.name_scope('input'):
  bottleneck_input = tf.placeholder_with_default(
      bottleneck_tensor,
      shape=[None, bottleneck_tensor_size],
      name='BottleneckInputPlaceholder')

  ground_truth_input = tf.placeholder(tf.float32,
                                      [None, class_count],
                                      name='GroundTruthInput')

# Organizing the following ops as `final_training_ops` so they're easier
# to see in TensorBoard
layer_name = 'final_training_ops'
with tf.name_scope(layer_name):
  with tf.name_scope('weights'):
    initial_value = tf.truncated_normal(
```

```
        [bottleneck_tensor_size, class_count], stddev=0.001)

    layer_weights = tf.Variable(initial_value, name='final_weights')

    variable_summaries(layer_weights)
  with tf.name_scope('biases'):
    layer_biases = tf.Variable(tf.zeros([class_count]), name='final_biases')
    variable_summaries(layer_biases)
  with tf.name_scope('Wx_plus_b'):
    logits = tf.matmul(bottleneck_input, layer_weights) + layer_biases
    tf.summary.histogram('pre_activations', logits)

final_tensor = tf.nn.softmax(logits, name=final_tensor_name)
tf.summary.histogram('activations', final_tensor)

with tf.name_scope('cross_entropy'):
  cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
      labels=ground_truth_input, logits=logits)
  with tf.name_scope('total'):
    cross_entropy_mean = tf.reduce_mean(cross_entropy)
tf.summary.scalar('cross_entropy', cross_entropy_mean)

with tf.name_scope('train'):
  optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate)
  train_step = optimizer.minimize(cross_entropy_mean)

return (train_step, cross_entropy_mean, bottleneck_input, ground_truth_input,
        final_tensor)


def add_evaluation_step(result_tensor, ground_truth_tensor):
 """Inserts the operations we need to evaluate the accuracy of our results.

 Args:
  result_tensor: The new final node that produces results.
  ground_truth_tensor: The node we feed ground truth data
```

```
  into.

 Returns:
  Tuple of (evaluation step, prediction).
 """
 with tf.name_scope('accuracy'):
  with tf.name_scope('correct_prediction'):
   prediction = tf.argmax(result_tensor, 1)
   correct_prediction = tf.equal(
       prediction, tf.argmax(ground_truth_tensor, 1))
  with tf.name_scope('accuracy'):
   evaluation_step = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
 tf.summary.scalar('accuracy', evaluation_step)
 return evaluation_step, prediction


def save_graph_to_file(sess, graph, graph_file_name):
 output_graph_def = graph_util.convert_variables_to_constants(
    sess, graph.as_graph_def(), [FLAGS.final_tensor_name])
 with gfile.FastGFile(graph_file_name, 'wb') as f:
  f.write(output_graph_def.SerializeToString())
 return


def prepare_file_system():
 # Setup the directory we'll write summaries to for TensorBoard
 if tf.gfile.Exists(FLAGS.summaries_dir):
  tf.gfile.DeleteRecursively(FLAGS.summaries_dir)
 tf.gfile.MakeDirs(FLAGS.summaries_dir)
 if FLAGS.intermediate_store_frequency > 0:
  ensure_dir_exists(FLAGS.intermediate_output_graphs_dir)
 return


def create_model_info(architecture):
 """Given the name of a model architecture, returns information about it.
```

```
  There are different base image recognition pretrained models that can be
  retrained using transfer learning, and this function translates from the name
  of a model to the attributes that are needed to download and train with it.

  Args:
    architecture: Name of a model architecture.

  Returns:
    Dictionary of information about the model, or None if the name isn't
    recognized

  Raises:
    ValueError: If architecture name is unknown.
  """
  architecture = architecture.lower()
  if architecture == 'inception_v3':
    # pylint: disable=line-too-long
    data_url = 'http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz'
    # pylint: enable=line-too-long
    bottleneck_tensor_name = 'pool_3/_reshape:0'
    bottleneck_tensor_size = 2048
    input_width = 299
    input_height = 299
    input_depth = 3
    resized_input_tensor_name = 'Mul:0'
    model_file_name = 'classify_image_graph_def.pb'
    input_mean = 128
    input_std = 128
  elif architecture.startswith('mobilenet_'):
    parts = architecture.split('_')
    if len(parts) != 3 and len(parts) != 4:
      tf.logging.error("Couldn't understand architecture name '%s'",
                       architecture)
      return None
    version_string = parts[1]
```

```
  if (version_string != '1.0' and version_string != '0.75' and
      version_string != '0.50' and version_string != '0.25'):
    tf.logging.error(
        """The Mobilenet version should be '1.0', '0.75', '0.50', or '0.25',
 but found '%s' for architecture '%s'""",
        version_string, architecture)
    return None
  size_string = parts[2]
  if (size_string != '224' and size_string != '192' and
      size_string != '160' and size_string != '128'):
    tf.logging.error(
        """The Mobilenet input size should be '224', '192', '160', or '128',
 but found '%s' for architecture '%s'""",
        size_string, architecture)
    return None
  if len(parts) == 3:
    is_quantized = False
  else:
    if parts[3] != 'quantized':
      tf.logging.error(
          "Couldn't understand architecture suffix '%s' for '%s'", parts[3],
          architecture)
      return None
    is_quantized = True
  data_url = 'http://download.tensorflow.org/models/mobilenet_v1_'
  data_url += version_string + '_' + size_string + '_frozen.tgz'
  bottleneck_tensor_name = 'MobilenetV1/Predictions/Reshape:0'
  bottleneck_tensor_size = 1001
  input_width = int(size_string)
  input_height = int(size_string)
  input_depth = 3
  resized_input_tensor_name = 'input:0'
  if is_quantized:
    model_base_name = 'quantized_graph.pb'
  else:
    model_base_name = 'frozen_graph.pb'
```

```
    model_dir_name = 'mobilenet_v1_' + version_string + '_' + size_string
    model_file_name = os.path.join(model_dir_name, model_base_name)
    input_mean = 127.5
    input_std = 127.5
  else:
    tf.logging.error("Couldn't understand architecture name '%s'", architecture)
    raise ValueError('Unknown architecture', architecture)

  return {
      'data_url': data_url,
      'bottleneck_tensor_name': bottleneck_tensor_name,
      'bottleneck_tensor_size': bottleneck_tensor_size,
      'input_width': input_width,
      'input_height': input_height,
      'input_depth': input_depth,
      'resized_input_tensor_name': resized_input_tensor_name,
      'model_file_name': model_file_name,
      'input_mean': input_mean,
      'input_std': input_std,
  }


def add_jpeg_decoding(input_width, input_height, input_depth, input_mean,
                      input_std):
  """Adds operations that perform JPEG decoding and resizing to the graph..

  Args:
    input_width: Desired width of the image fed into the recognizer graph.
    input_height: Desired width of the image fed into the recognizer graph.
    input_depth: Desired channels of the image fed into the recognizer graph.
    input_mean: Pixel value that should be zero in the image for the graph.
    input_std: How much to divide the pixel values by before recognition.

  Returns:
    Tensors for the node to feed JPEG data into, and the output of the
      preprocessing steps.
```

```
  """
  jpeg_data = tf.placeholder(tf.string, name='DecodeJPGInput')
  decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_depth)
  decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
  decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
  resize_shape = tf.stack([input_height, input_width])
  resize_shape_as_int = tf.cast(resize_shape, dtype=tf.int32)
  resized_image = tf.image.resize_bilinear(decoded_image_4d,
                                           resize_shape_as_int)
  offset_image = tf.subtract(resized_image, input_mean)
  mul_image = tf.multiply(offset_image, 1.0 / input_std)
  return jpeg_data, mul_image


def main(_):
  # Needed to make sure the logging output is visible.
  # See https://github.com/tensorflow/tensorflow/issues/3047
  tf.logging.set_verbosity(tf.logging.INFO)

  # Prepare necessary directories  that can be used during training
  prepare_file_system()

  # Gather information about the model architecture we'll be using.
  model_info = create_model_info(FLAGS.architecture)
  if not model_info:
    tf.logging.error('Did not recognize architecture flag')
    return -1

  # Set up the pre-trained graph.
  maybe_download_and_extract(model_info['data_url'])
  graph, bottleneck_tensor, resized_image_tensor = (
      create_model_graph(model_info))

  # Look at the folder structure, and create lists of all the images.
  image_lists = create_image_lists(FLAGS.image_dir, FLAGS.testing_percentage,
                                   FLAGS.validation_percentage)
```

```
class_count = len(image_lists.keys())
if class_count == 0:
  tf.logging.error('No valid folders of images found at ' + FLAGS.image_dir)
  return -1
if class_count == 1:
  tf.logging.error('Only one valid folder of images found at ' +
                   FLAGS.image_dir +
                   ' - multiple classes are needed for classification.')
  return -1

# See if the command-line flags mean we're applying any distortions.
do_distort_images = should_distort_images(
    FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale,
    FLAGS.random_brightness)

with tf.Session(graph=graph) as sess:
  # Set up the image decoding sub-graph.
  jpeg_data_tensor, decoded_image_tensor = add_jpeg_decoding(
      model_info['input_width'], model_info['input_height'],
      model_info['input_depth'], model_info['input_mean'],
      model_info['input_std'])

  if do_distort_images:
    # We will be applying distortions, so setup the operations we'll need.
    (distorted_jpeg_data_tensor,
     distorted_image_tensor) = add_input_distortions(
         FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale,
         FLAGS.random_brightness, model_info['input_width'],
         model_info['input_height'], model_info['input_depth'],
         model_info['input_mean'], model_info['input_std'])
  else:
    # We'll make sure we've calculated the 'bottleneck' image summaries and
    # cached them on disk.
    cache_bottlenecks(sess, image_lists, FLAGS.image_dir,
                      FLAGS.bottleneck_dir, jpeg_data_tensor,
                      decoded_image_tensor, resized_image_tensor,
```

```
                bottleneck_tensor, FLAGS.architecture)

# Add the new layer that we'll be training.
(train_step, cross_entropy, bottleneck_input, ground_truth_input,
 final_tensor) = add_final_training_ops(
    len(image_lists.keys()), FLAGS.final_tensor_name, bottleneck_tensor,
    model_info['bottleneck_tensor_size'])

# Create the operations we need to evaluate the accuracy of our new layer.
evaluation_step, prediction = add_evaluation_step(
   final_tensor, ground_truth_input)

# Merge all the summaries and write them out to the summaries_dir
merged = tf.summary.merge_all()
train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train',
                                     sess.graph)

validation_writer = tf.summary.FileWriter(
   FLAGS.summaries_dir + '/validation')

# Set up all our weights to their initial default values.
init = tf.global_variables_initializer()
sess.run(init)

# Run the training for as many cycles as requested on the command line.
for i in range(FLAGS.how_many_training_steps):
  # Get a batch of input bottleneck values, either calculated fresh every
  # time with distortions applied, or from the cache stored on disk.
  if do_distort_images:
    (train_bottlenecks,
     train_ground_truth) = get_random_distorted_bottlenecks(
        sess, image_lists, FLAGS.train_batch_size, 'training',
        FLAGS.image_dir, distorted_jpeg_data_tensor,
        distorted_image_tensor, resized_image_tensor, bottleneck_tensor)
  else:
    (train_bottlenecks,
```

```
    train_ground_truth, _) = get_random_cached_bottlenecks(
        sess, image_lists, FLAGS.train_batch_size, 'training',
        FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
        decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
        FLAGS.architecture)
    # Feed the bottlenecks and ground truth into the graph, and run a training
    # step. Capture training summaries for TensorBoard with the `merged` op.
    train_summary, _ = sess.run(
      [merged, train_step],
      feed_dict={bottleneck_input: train_bottlenecks,
                 ground_truth_input: train_ground_truth})
    train_writer.add_summary(train_summary, i)

    # Every so often, print out how well the graph is training.
    is_last_step = (i + 1 == FLAGS.how_many_training_steps)
    if (i % FLAGS.eval_step_interval) == 0 or is_last_step:
      train_accuracy, cross_entropy_value = sess.run(
          [evaluation_step, cross_entropy],
          feed_dict={bottleneck_input: train_bottlenecks,
                     ground_truth_input: train_ground_truth})
      tf.logging.info('%s: Step %d: Train accuracy = %.1f%%' %
                      (datetime.now(), i, train_accuracy * 100))
      tf.logging.info('%s: Step %d: Cross entropy = %f' %
                      (datetime.now(), i, cross_entropy_value))
      validation_bottlenecks, validation_ground_truth, _ = (
          get_random_cached_bottlenecks(
              sess, image_lists, FLAGS.validation_batch_size, 'validation',
              FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
              decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
              FLAGS.architecture))
      # Run a validation step and capture training summaries for TensorBoard
      # with the `merged` op.
      validation_summary, validation_accuracy = sess.run(
          [merged, evaluation_step],
          feed_dict={bottleneck_input: validation_bottlenecks,
                     ground_truth_input: validation_ground_truth})
```

```
    validation_writer.add_summary(validation_summary, i)
    tf.logging.info('%s: Step %d: Validation accuracy = %.1f%% (N=%d)' %
                    (datetime.now(), i, validation_accuracy * 100,
                     len(validation_bottlenecks)))

  # Store intermediate results
  intermediate_frequency = FLAGS.intermediate_store_frequency

  if (intermediate_frequency > 0 and (i % intermediate_frequency == 0)
      and i > 0):
    intermediate_file_name = (FLAGS.intermediate_output_graphs_dir +
                              'intermediate_' + str(i) + '.pb')
    tf.logging.info('Save intermediate result to : ' +
                    intermediate_file_name)
    save_graph_to_file(sess, graph, intermediate_file_name)

# We've completed all our training, so run a final test evaluation on
# some new images we haven't used before.
test_bottlenecks, test_ground_truth, test_filenames = (
    get_random_cached_bottlenecks(
        sess, image_lists, FLAGS.test_batch_size, 'testing',
        FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
        decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
        FLAGS.architecture))
test_accuracy, predictions = sess.run(
    [evaluation_step, prediction],
    feed_dict={bottleneck_input: test_bottlenecks,
               ground_truth_input: test_ground_truth})
tf.logging.info('Final test accuracy = %.1f%% (N=%d)' %
                (test_accuracy * 100, len(test_bottlenecks)))

if FLAGS.print_misclassified_test_images:
  tf.logging.info('=== MISCLASSIFIED TEST IMAGES ===')
  for i, test_filename in enumerate(test_filenames):
    if predictions[i] != test_ground_truth[i].argmax():
      tf.logging.info('%70s  %s' %
```

```
                (test_filename,
                 list(image_lists.keys())[predictions[i]]))

  # Write out the trained graph and labels with the weights stored as
  # constants.
  save_graph_to_file(sess, graph, FLAGS.output_graph)
  with gfile.FastGFile(FLAGS.output_labels, 'w') as f:
    f.write('\n'.join(image_lists.keys()) + '\n')


if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--image_dir',
      type=str,
      default='',
      help='Path to folders of labeled images.'
  )
  parser.add_argument(
      '--output_graph',
      type=str,
      default='/tmp/output_graph.pb',
      help='Where to save the trained graph.'
  )
  parser.add_argument(
      '--intermediate_output_graphs_dir',
      type=str,
      default='/tmp/intermediate_graph/',
      help='Where to save the intermediate graphs.'
  )
  parser.add_argument(
      '--intermediate_store_frequency',
      type=int,
      default=0,
      help="""\
        How many steps to store intermediate graph. If "0" then will not
```

```
      store.\
    """
)
parser.add_argument(
    '--output_labels',
    type=str,
    default='/tmp/output_labels.txt',
    help='Where to save the trained graph\'s labels.'
)
parser.add_argument(
    '--summaries_dir',
    type=str,
    default='/tmp/retrain_logs',
    help='Where to save summary logs for TensorBoard.'
)
parser.add_argument(
    '--how_many_training_steps',
    type=int,
    default=6000,
    help='How many training steps to run before ending.'
)
parser.add_argument(
    '--learning_rate',
    type=float,
    default=0.01,
    help='How large a learning rate to use when training.'
)
parser.add_argument(
    '--testing_percentage',
    type=int,
    default=10,
    help='What percentage of images to use as a test set.'
)
parser.add_argument(
    '--validation_percentage',
    type=int,
```

```
    default=10,
    help='What percentage of images to use as a validation set.'
)
parser.add_argument(
    '--eval_step_interval',
    type=int,
    default=10,
    help='How often to evaluate the training results.'
)
parser.add_argument(
    '--train_batch_size',
    type=int,
    default=100,
    help='How many images to train on at a time.'
)
parser.add_argument(
    '--test_batch_size',
    type=int,
    default=-1,
    help="""\
    How many images to test on. This test set is only used once, to evaluate
    the final accuracy of the model after training completes.
    A value of -1 causes the entire test set to be used, which leads to more
    stable results across runs.\
    """
)
parser.add_argument(
    '--validation_batch_size',
    type=int,
    default=100,
    help="""\
    How many images to use in an evaluation batch. This validation set is
    used much more often than the test set, and is an early indicator of how
    accurate the model is during training.
    A value of -1 causes the entire validation set to be used, which leads to
    more stable results across training iterations, but may be slower on large
```

```
    training sets.\
    """
)
parser.add_argument(
    '--print_misclassified_test_images',
    default=False,
    help="""\
    Whether to print out a list of all misclassified test images.\
    """,
    action='store_true'
)
parser.add_argument(
    '--model_dir',
    type=str,
    default='/tmp/imagenet',
    help="""\
    Path to classify_image_graph_def.pb,
    imagenet_synset_to_human_label_map.txt, and
    imagenet_2012_challenge_label_map_proto.pbtxt.\
    """
)
parser.add_argument(
    '--bottleneck_dir',
    type=str,
    default='/tmp/bottleneck',
    help='Path to cache bottleneck layer values as files.'
)
parser.add_argument(
    '--final_tensor_name',
    type=str,
    default='final_result',
    help="""\
    The name of the output classification layer in the retrained graph.\
    """
)
parser.add_argument(
```

```
    '--flip_left_right',
    default=False,
    help="""\
    Whether to randomly flip half of the training images horizontally.\
    """,
    action='store_true'
)
parser.add_argument(
    '--random_crop',
    type=int,
    default=0,
    help="""\
    A percentage determining how much of a margin to randomly crop off the
    training images.\
    """
)
parser.add_argument(
    '--random_scale',
    type=int,
    default=0,
    help="""\
    A percentage determining how much to randomly scale up the size of the
    training images by.\
    """
)
parser.add_argument(
    '--random_brightness',
    type=int,
    default=0,
    help="""\
    A percentage determining how much to randomly multiply the training image
    input pixels up or down by.\
    """
)
parser.add_argument(
    '--architecture',
```

```
    type=str,
    default='inception_v3',
    help="""\
    Which model architecture to use. 'inception_v3' is the most accurate, but
    also the slowest. For faster or smaller models, chose a MobileNet with the
    form 'mobilenet_<parameter size>_<input_size>[_quantized]'. For example,
    'mobilenet_1.0_224' will pick a model that is 17 MB in size and takes 224
    pixel input images, while 'mobilenet_0.25_128_quantized' will choose a much
    less accurate, but smaller and faster network that's 920 KB on disk and
    takes 128x128 images. See https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html
    for more information on Mobilenet.\
    """)
 FLAGS, unparsed = parser.parse_known_args()
 tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

TESTING.PY

```
import cv2
import numpy as np

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('TrainingImageLabel/trainner.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath)
font = cv2.FONT_HERSHEY_SIMPLEX

cam = cv2.VideoCapture(0)
while True:
   ret, im =cam.read()
   gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
   faces=faceCascade.detectMultiScale(gray, 1.2,5)
   for(x,y,w,h) in faces:
      Id, conf = recognizer.predict(gray[y:y+h,x:x+w])

      # # else:
      # #    Id="Unknown"
```

```
        # cv2.rectangle(im, (x-22,y-90), (x+w+22, y-22), (0,255,0), -1)
        cv2.rectangle(im, (x, y), (x + w, y + h), (0, 260, 0), 7)
        cv2.putText(im, str(Id), (x,y-40),font, 2, (255,255,255), 3)

        # cv2.putText(im, str(Id), (x + h, y), font, 1, (0, 260, 0), 2)
    cv2.imshow('im',im)
    if cv2.waitKey(10) & 0xFF==ord('q'):
        break
cam.release()
cv2.destroyAllWindows()
```

Training.py

```
import cv2,os
import numpy as np
from PIL import Image
#
# recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer=cv2.face.createFisherFaceRecognizer_create()
detector= cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

def getImagesAndLabels(path):
    #get the path of all the files in the folder
    imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
    #create empth face list
    faceSamples=[]
    #create empty ID list
    Ids=[]
    #now looping through all the image paths and loading the Ids and the images
    for imagePath in imagePaths:
        #loading the image and converting it to gray scale
        pilImage=Image.open(imagePath).convert('L')
        #Now we are converting the PIL image into numpy array
        imageNp=np.array(pilImage,'uint8')
        #getting the Id from the image
```

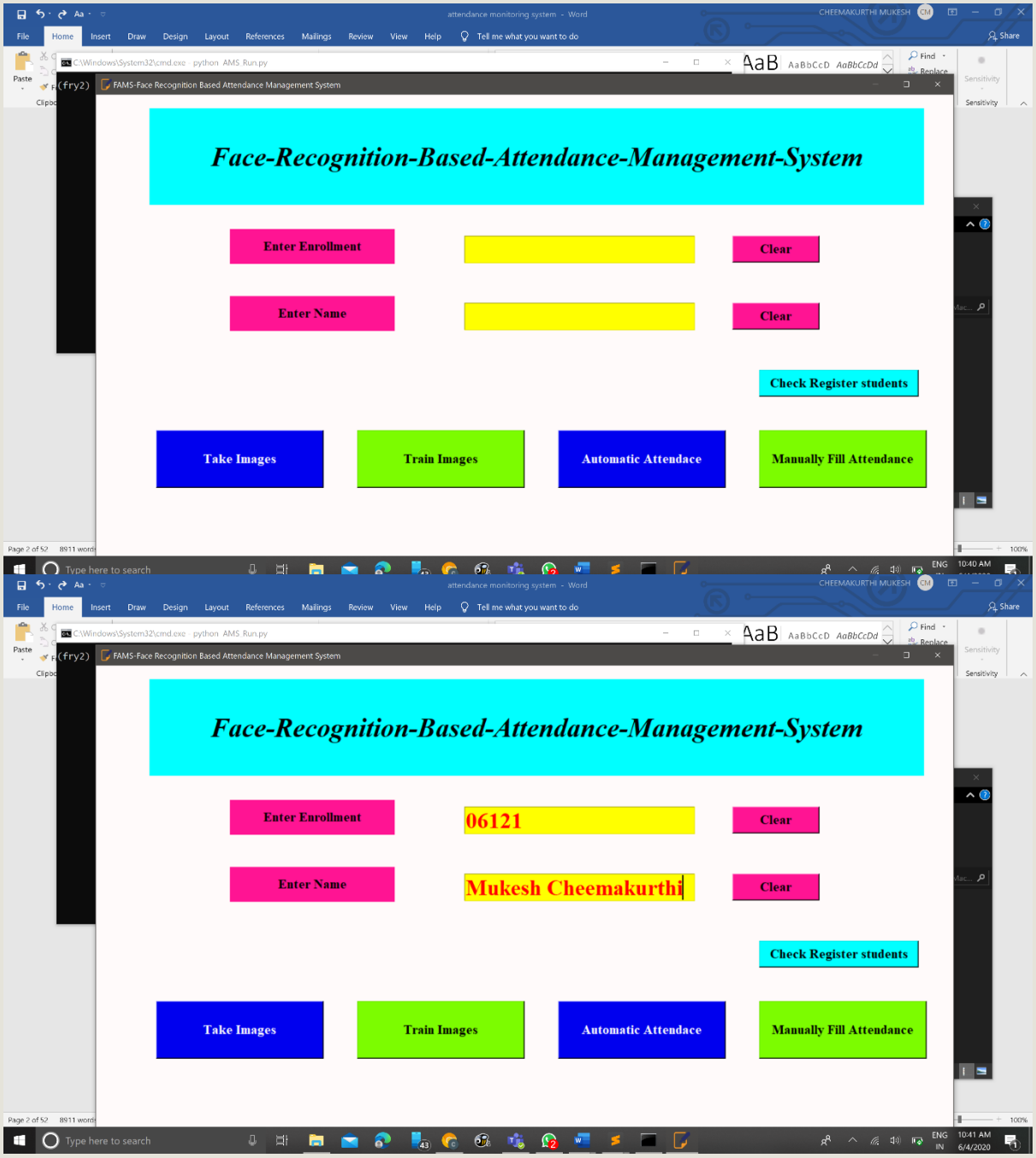
```
        Id = int(os.path.split(imagePath)[-1].split(".")[1])
        # extract the face from the training image sample
        faces=detector.detectMultiScale(imageNp)
        #If a face is there then append that in the list as well as Id of it
        for (x,y,w,h) in faces:
            faceSamples.append(imageNp[y:y+h,x:x+w])
            Ids.append(Id)
    return faceSamples,Ids

faces,Ids = getImagesAndLabels('TrainingImage')
recognizer.train(faces, np.array(Ids))
recognizer.save('TrainingImageLabel/trainner.yml')
```

7.OUTPUT AND PERFORMANCE ANALYSIS :

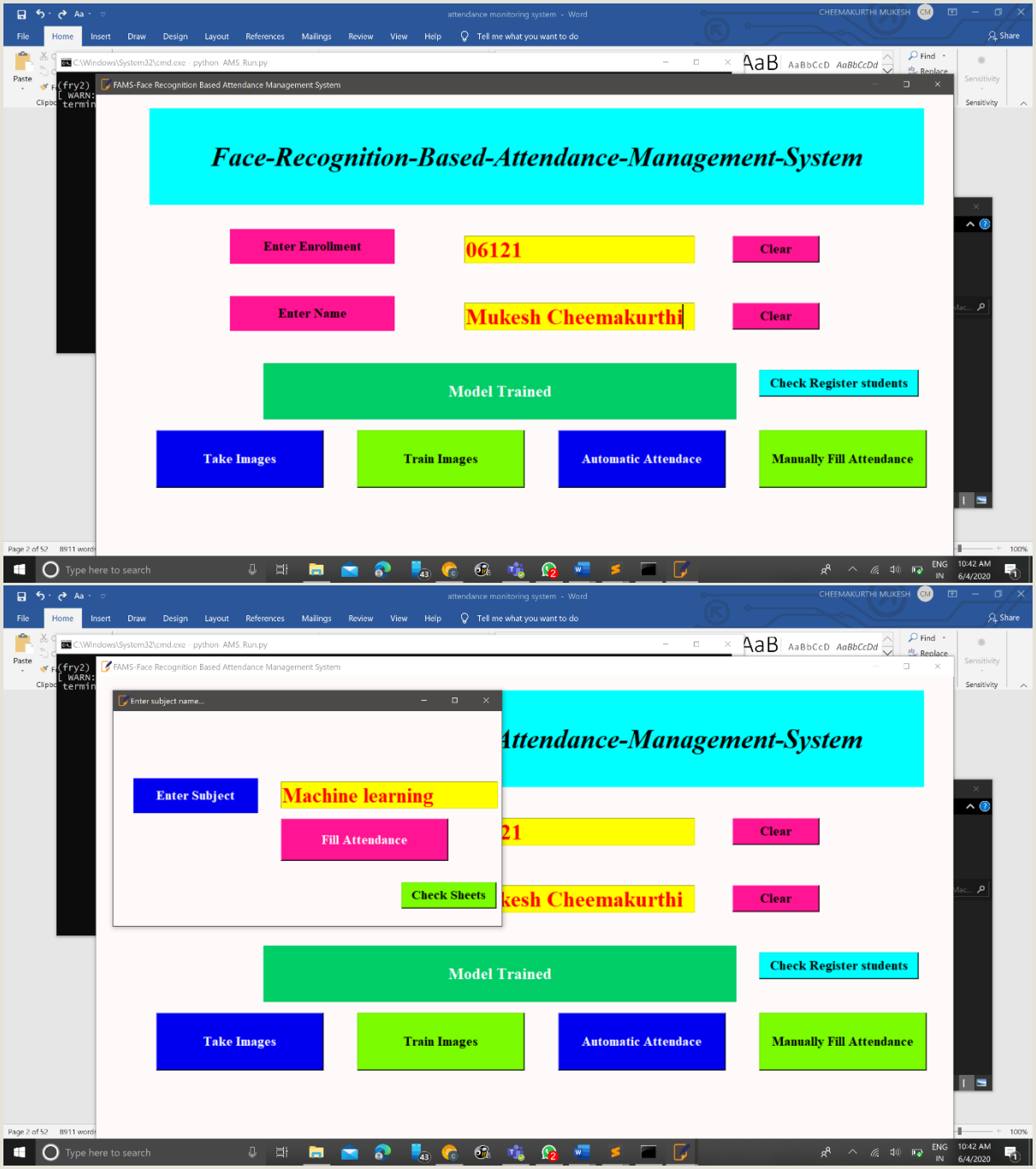
7.1 EXECUTION SNAP SHOTS :

-->go back to index page

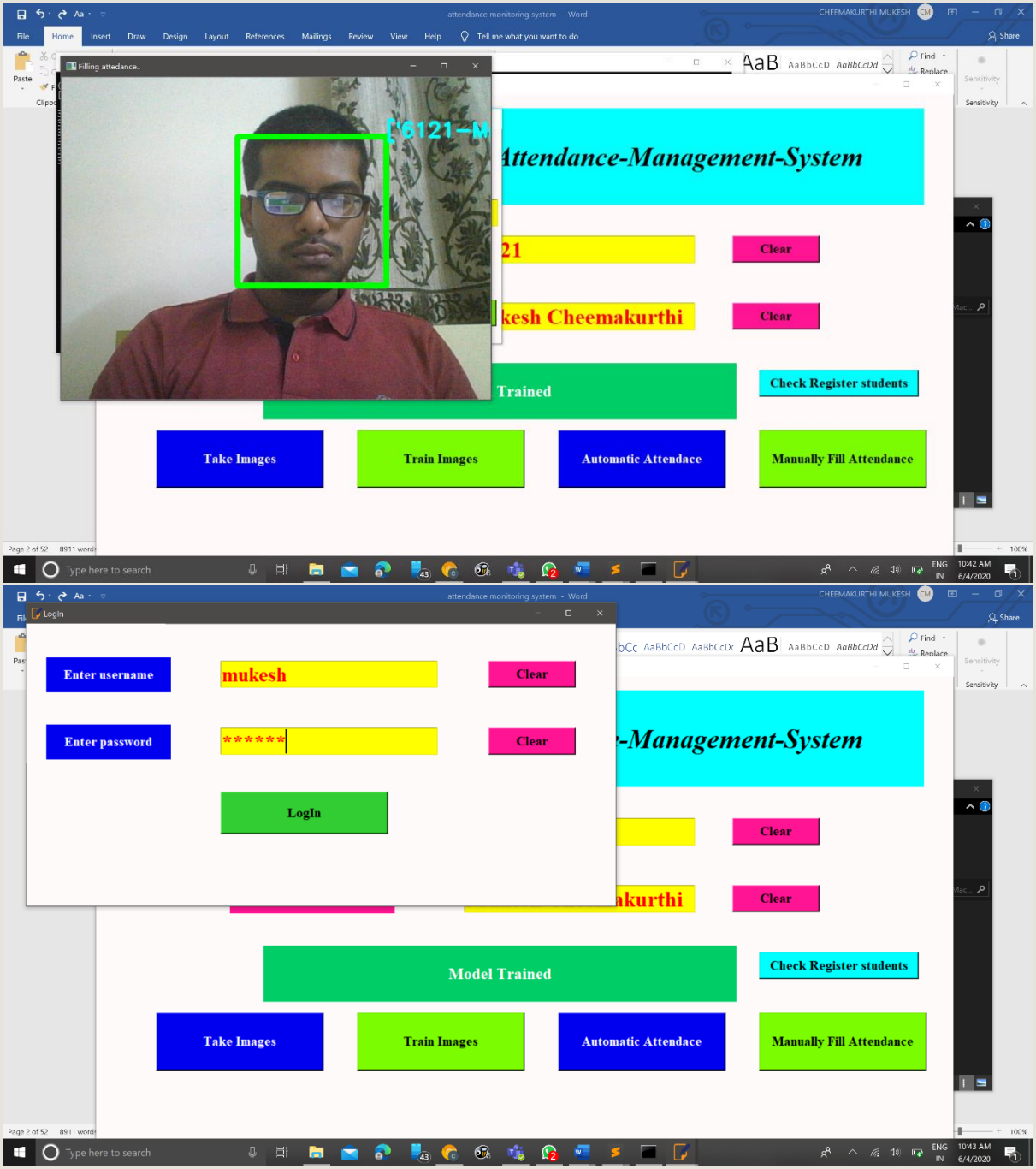


-->go back to index page

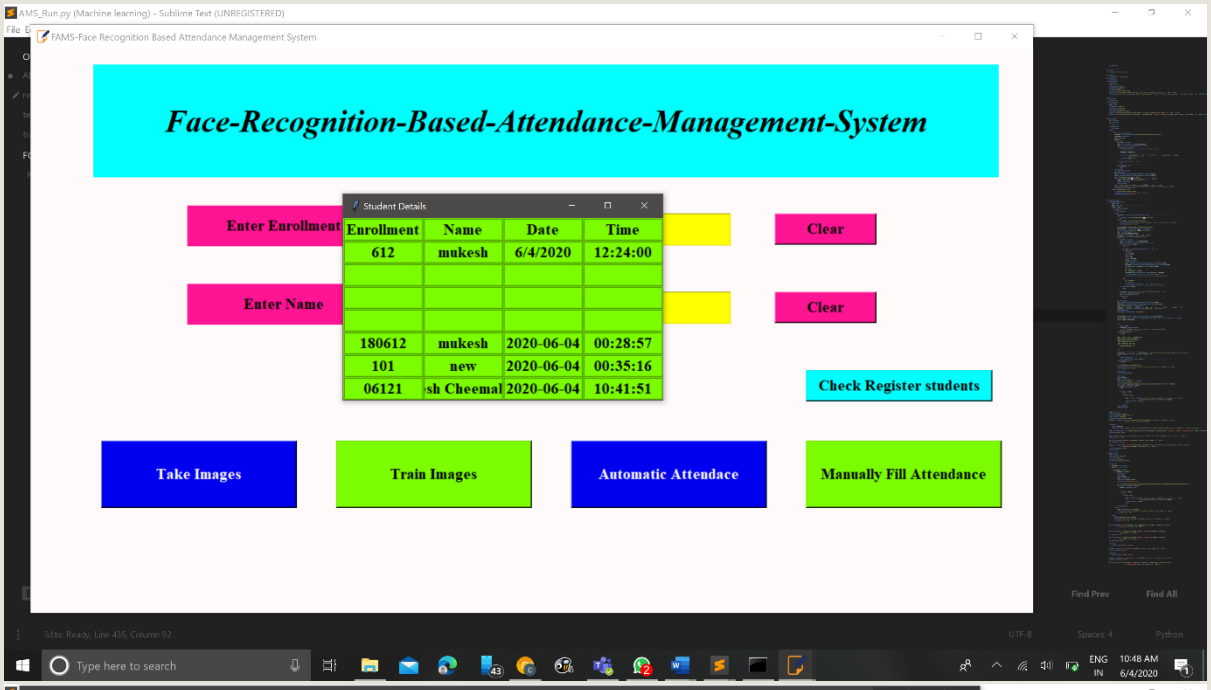
-->go back to index page

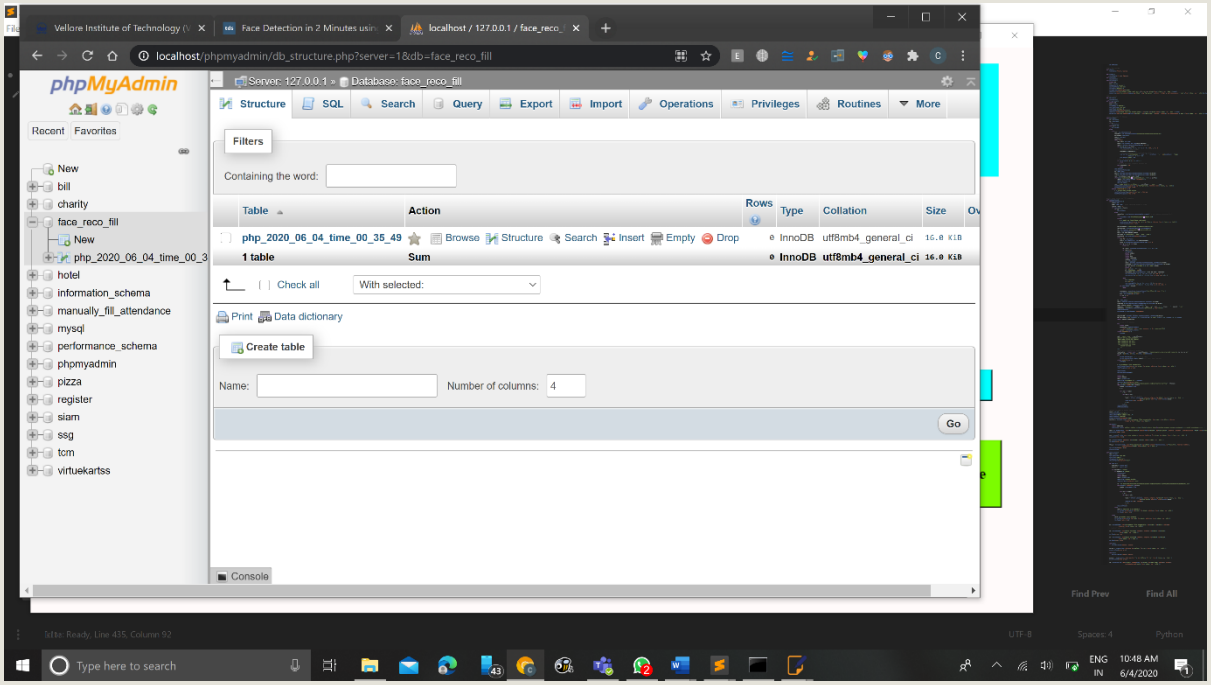


[-->go back to index page]

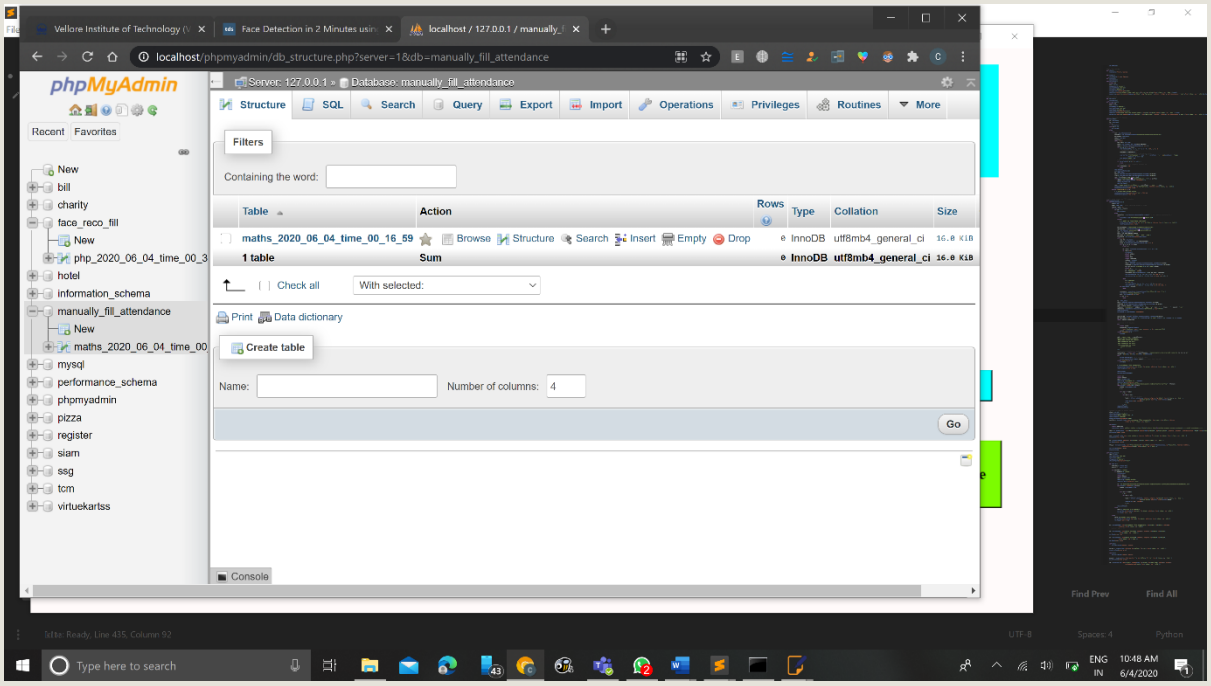


-->go back to index page





-->go back to index page



8. FUTURE DIRECTIVES :

This model can be used in automatic attendance filling using face recognition which supports multi face attendance marking. This can drastically improve the amount of time saved in class which is generally used for biometric or manual attendance . This model can be incorporated with cc cameras and directly mark attendance with out student or teacher's knowledge. In case of model failure or any bad detection this project can work inefficiently. So to handle this manual attendance system is also used. Data is stored in database(mysql) and in form of a file(.csv) . In this way it makes lives easier.

9.REFERENCES :

https://ieeexplore.ieee.org/abstract/document/7006273/

https://ieeexplore.ieee.org/abstract/document/8203730/

https://pdfs.semanticscholar.org/11ed/43556b09ef33a43ca52c8f22c3e1044be943.pdf