Neural Network Project Report

Submitted by

Aarushi Sood, Abhijeet Baruah & Keshav Tangri
[CO17301, CO17302, CO17333]

Submitted to

Dr. Varun Gupta

In partial fulfilment for the award of the degree
Of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY, SECTOR-26
CHANDIGARH-160019
PANJAB UNIVERSITY, CHANDIGARH

EyeAttend – Facial Recognition based Attendance System from scratch.

Keshav Tangri
Department of Computer Science and Engineering
Chandigarh College of Engineering and Technology
tangri57@gmail.com
CO17333

Abhijeet Baruah
Department of Computer Science and Engineering
Chandigarh College of Engineering and Technology
abhibaruah4297@gmail.com
CO17302

Aarushi Sood
Department of Computer Science and Engineering
Chandigarh College of Engineering and Technology
aarushisood1010@gmail.com
CO17301

Abstract – Facial Recognition follows a biometric identification methodology and is a technology capable of object verification or identification based on the biometric patterns of facial region. Input can be given as any audio-visual element such as image, video etc and the facial patterns helps to give access to the concerned user. [1] In this project, we have extended the application to a day to day activity in the life of students, i.e. the Attendance Management. Our proposal is for an intelligent system that can harness the power of mobile devices and deep learning in order to automate the process of attendance generation in a quick and hassle free way. We have developed an Android App that will be the interface of the whole project.

Keywords – Deep Learning, Facial Recognition, Native Android App, Convolutional Neural Networks, Attendance System.

I. INTRODUCTION

The project is our final year's mini project and we have chosen the domain to be Artificial Intelligence. Digging deep inside artificial intelligence lies the domain of Machine Learning and we have chosen our base to be Deep Learning (subset of ML).

We remember that during our first year of college, we felt a need of something inspiring, something that would amaze us and hence this thought became the motivation for our project. Following this motivation, we decided to give back what we thought at that time, almost after four years of B.E. in CSE via this project.

Our vision for this project is to introduce an intelligent system for motivating freshman year students in machine learning. The aim is to deploy this system in our college so that, once the freshman students observe its application in their own lives, this would definitely ignite the fire in some students to build a better version, better projects that can be used for socio-eco platforms.

The target is easy as every class has it's own class representative and one of the main duty is taking the attendance of students. In this way we can have the direct interaction with the target audience i.e. the fresh minds of college.

We have the following takeaways from this project

1. A scalable design with the focus on making the attendance system not only automatic but also intelligent.

2. The architecture of the system mainly focuses on its usability and scalability which will easily help us to redesign it in accordance to the future feedbacks.

3. With a modular approach, ML models can be plug in/plug out.

4. Use machine learning and vectorization for faster results.

5. Android Application- "EyeAttend" for interacting with the system.

Before beginning with a solution to any problem, it is always advised to understand the problem first. Therefore, we begin by explaining the current/traditional methodology being undertaken for taking attendance.

II. TRADITIONAL METHODOLOGY

1. Initially when a lecture ends, the class representative will have to stand by the podium and call out the roll number of each student and the concerned individual will revert with present if he/she is in the class.

2. Next the class representative (CR) will take a note of the present and absent and/or absent students.

3. This list is then sent to concerned subject teacher. The authority to fill attendance is not given to the class representative.

4. The teacher keeps note of this in his/her register.

5. This cycle is repeated till the end of semester

III. LIMITATIONS OF TRADITIONAL METHODOLOGY

There are many limitations involved with this particular mechanism. Few of them are as follows:
1. Proxy: Proxy means Fake attendance in lectures in colleges. The above methodology is prone to proxy at the CR's level. He might mark present for his friends even though they are roaming out in the city.

2. Sharing: It is difficult to share the attendance with the students. Sometimes it is due to no updation in the register, or the professor is unavailable etc. In countries like India, where 75% attendance is compulsory in colleges, this is a big issue for the students.

3. No Softcopy: If the attendance sheet is to be forwarded after the end of the semester, it is difficult to do so as there is no softcopy maintained. This results into another time wastage for scanning the register.

4. Extensive Manual Work: Whether it is updating attendance on an excel or a register, the traditional methodology involves a lot of manual effort for accomplishing the task.

5. Time: The whole process is somewhat time consuming.

6. Editing: In case of an absentee due to illness, some competition or any reason which has a valid proof and has to be converted back to present, the professor has to use whitener to correct it which makes the register kind of messy.

III. OUR PROPOSAL

Our proposal is a hassle free deployment of a system which do not require the need of CR to call out names and the teacher itself can take the attendance by just clicking a photo of the students. We first upload our image to the database, next the machine learning model extracts faces from the image and then based on the teacher, subject code, and batch facial recognition is performed between the extracted face and the classroom faces. Once this is done, the presentees and absentees list is generated and gets dynamically updated on the DataBase. This final list is sent back to the front end for the teacher. The solution also provides easy download facility and in future versions, we can add graphical analytics of the data collected to visually understand trends over the year.

A. Flow of the project (ML Model flow)

We have segregated models with different functionality for easy future updation and scalability.

The below flow shows how our image is processed by the various models. Initially we thought of making the project for both masked and unmasked scenarios, but due to hardware constrains we took the path of unmasked only. However the new models can be developed and the code can be updated in order to incorporate Masked version too.

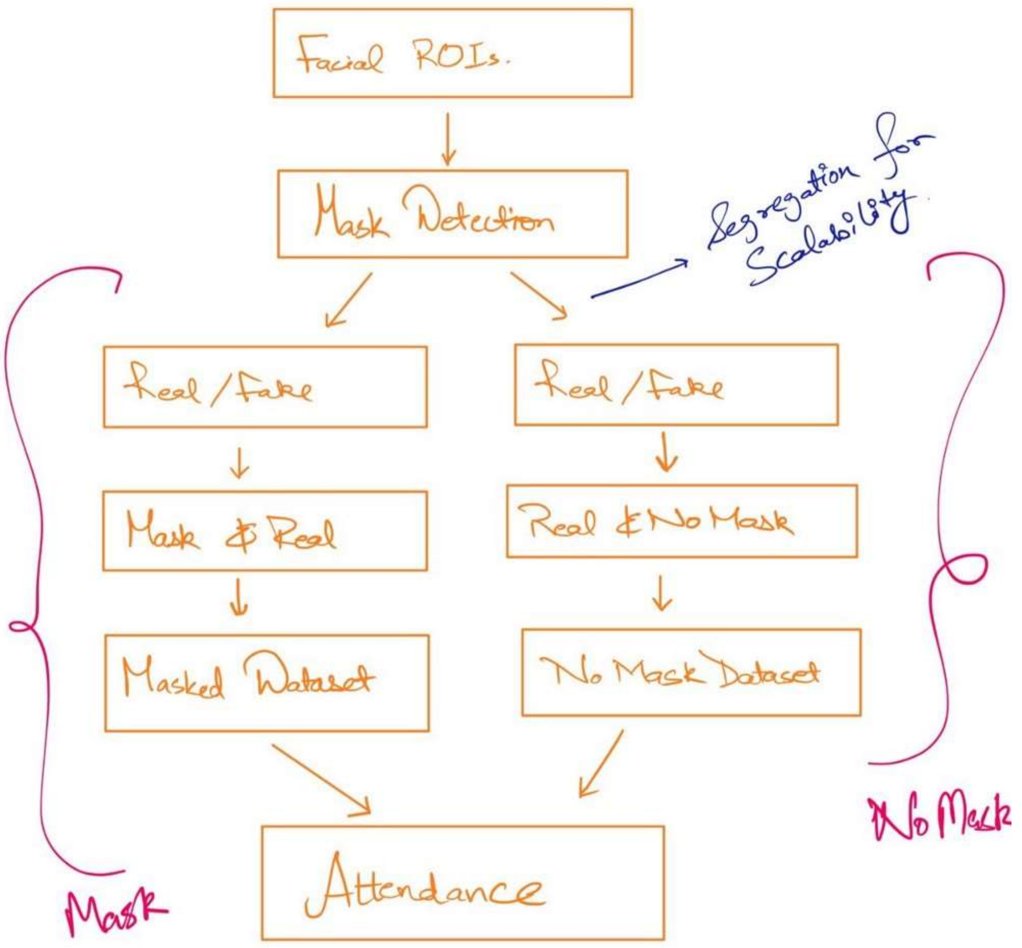
So, initially when the image is given to the script, it will extract the facial regions from it. It will detect the presence of faces and will extract it in 224x224x3 dimensions. Next we iterate over the number of faces and for each face, we do the following :

1. Check if it is masked or unmasked
2. Check if it is a real person or a picture of person's face

3. Check who is the person from the classroom (facial recognition).

4. Mark his roll number and keep him in presentees record.

i. Models Overview



As mentioned in the above section, we have to develop different models for our model. Now this is because as the new students arrive in campus, we might need to train the model again to make it more robust. This is the reason why the models are broken down into different segments so that one segment can be upgraded and then reused without affecting the other models. Listed below are the models which we have used and which we are going to develop:

1. keras-facenet (Transfer Learning): This is used for extracting faces and creation of embeddings of face images[2]. We have used a pretrained facenet model that converts a face image into 512 dimensional vector. It detects and extracts faces via the MTCNN (Multi Task Cascaded Neural Networks).

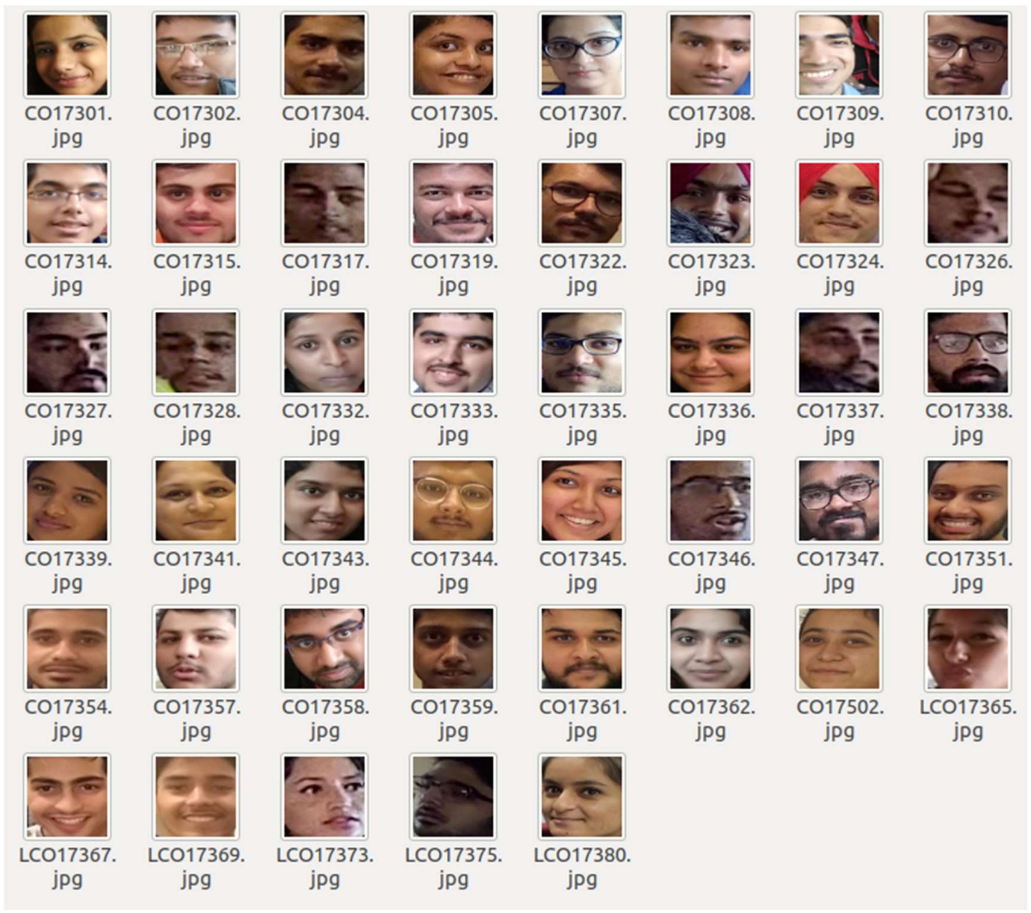
2. mask-detector: We will be developing this model to detect whether the person's face has a mask or not. This is done to separate the different use case and with mask, the classroom database, real/fake detector etc will be developed differently.

3. real/fake detector: The real/fake detector or liveliness detector model is used to check if the actual human is present. This minimizes proxy in our project as it might be possible that someone just opens up the laptop screen with their friend's face image so that his/her roll number can be counted in presentees.

B. Creating Classroom Batch in DataBase.

In order to perform the Recognition, we must have the class dataset with us. In our case, since the universities were closed, so we had to search our own gallery and look for group images or single images of our friends in order to make the classroom dataset. Once we have a collection of photos, we ran them through the MTCNN (Multi Task Cascaded Neural Networks) via the keras_facenet pretrained model package, for detecting the faces in the images and extracting the bounding areas of the faces.

Next we selected the best possible images for our classmates and created a folder and saved the data named by their roll numbers in the folder. The data after collection and segregation looks as shown below :

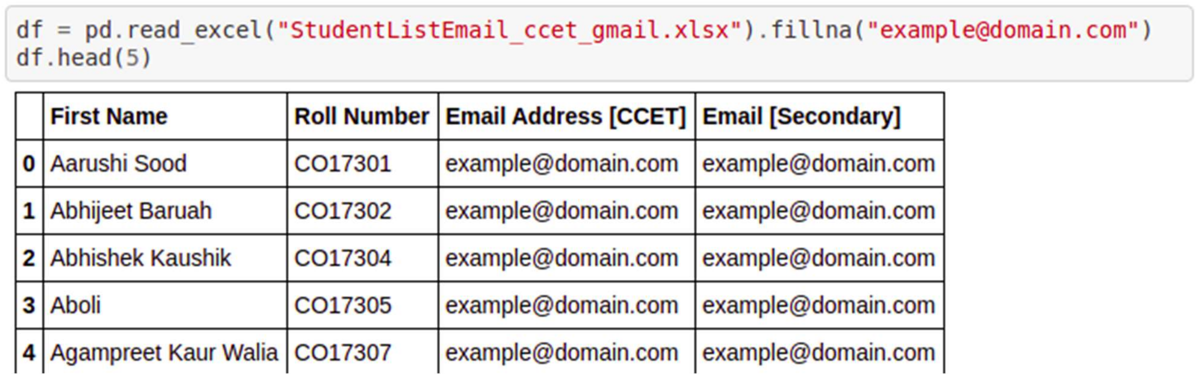


The above folder consists of student from CSE branch, batch 2017-21.

Next we need to upload the same into our Database, along with their details, such as name, roll number, email etc.

For this, we developed a python script to upload the same. Every Batch, when arrives at college, an excel sheet is maintained by the Class Representative having the details of each and every student. We utilized that data for the same. We had an excel sheet for this batch, sorted by their Roll Numbers (like the face images above). We kept only those roll numbers in the excel for which we had the images. OfCourse when deployed in College, this won't be the case as we will have students available and missing photographs could be directly taken in the college premises at the required time.

The excel code snippet is shown below :



```
df = pd.read_excel("StudentListEmail_ccet_gmail.xlsx").fillna("example@domain.com")
df.head(5)
```

	First Name	Roll Number	Email Address [CCET]	Email [Secondary]
0	Aarushi Sood	CO17301	example@domain.com	example@domain.com
1	Abhijeet Baruah	CO17302	example@domain.com	example@domain.com
2	Abhishek Kaushik	CO17304	example@domain.com	example@domain.com
3	Aboli	CO17305	example@domain.com	example@domain.com
4	Agampreet Kaur Walia	CO17307	example@domain.com	example@domain.com

For saving data from theft, we have currently added email as example@domain.com. However in our project actual email is used (for authorization).

So the python code for the same is shown below :

```
import os
# os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
from PIL import Image
from keras.models import load_model
from keras_facenet import FaceNet
import numpy as np
from mtcnn import MTCNN
import cv2
import pickle
from IPython.display import display
import pandas as pd

from tensorflow.keras.preprocessing.image import img_to_array,array_to_img
embedder = FaceNet()
import mysql.connector
```

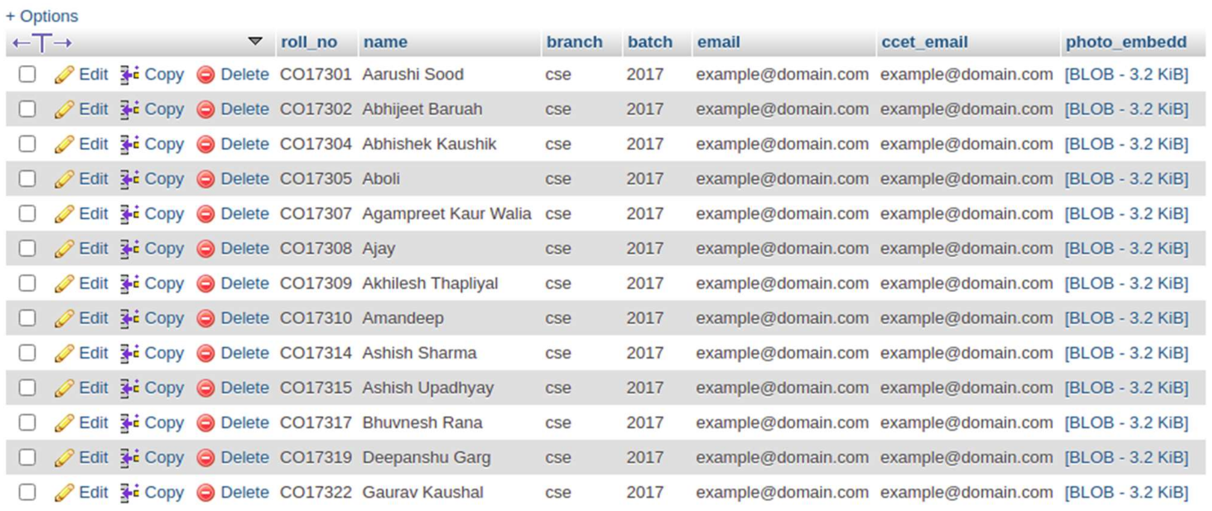
```
df = pd.read_excel("StudentListEmail_ccet_gmail.xlsx").fillna("example@domain.com")
print(df.head(5))

name = list(df['First Name '])
email = list(df['Email [Secondary]'])
ccet_email = list(df['Email Address [CCET]'])
roll= list(df['Roll Number'])

class_temp_path ="/home/zukayu/Eye_Attend/Eye_Attend_Final/classroom_224"

temp_totalEmbeddings = []
for faces in os.listdir(class_temp_path):
    face = Image.open(class_temp_path+os.sep+faces)
    face = img_to_array(face)
    face = np.expand_dims(face, axis=0)
    embedding = embedder.embeddings(face)
    temp_totalEmbeddings.append(embedding)

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="eyeattend"
    )
if conn.is_connected():
    print("Successfully connected")
    mycursor = conn.cursor()
    #sql = "INSERT INTO `embeddings`(`embedd`) VALUES (%s)"
    #"INSERT INTO table VALUES (%s, %s, %s)", (pickles)
    create_tab = "CREATE TABLE IF NOT EXISTS `batch_2017` ( `roll_no` varchar(10) NOT
NULL, \
                `name` varchar(50) NOT NULL,\
                `branch` text NOT NULL,\
                `batch` text NOT NULL,\
                `email` varchar(50) NOT NULL,\
                `ccet_email` varchar(50) NOT NULL,\
                `photo_embedd` longblob NOT NULL,\
                PRIMARY KEY (`roll_no`) ) "

    mycursor.execute(create_tab)
    conn.commit()

    for i in range(len(roll)):
        r = str(roll[i])
        naam = str(name[i])
        ema = str(email[i])
        ccet = str(ccet_email[i])
        embed = temp_totalEmbeddings[i]
        pickemb = embed.dumps()
        mycursor.execute("INSERT INTO batch_2017 VALUES
(%s,%s,%s,%s,%s,%s,%s)",(r,naam,'cse','2017',ema,ccet,pickemb))
        conn.commit()
        # Closing the connection
    conn.close()

else:
    print('Error in connecting with database')
```

Explanation : The code starts by importing the required libraries. Next we open the excel file using pandas. We store the details of each student in respective lists (name, email, ccet_email, roll) and next we iterated over the images in the classroom folder. For each face image, we convert it to numpy array and put it in the pretrained Facenet Embedder to create an embedding of the same. Next we append this embedding in a list.

This embedding is nothing but a 512 – dimensional vector as previously mentioned in the Models Overview section. Each face is converted to 1 x 512 dimensional vector.

Next we make a connection variable named "conn", to connect with the backend MySQL Database. If the connection is successful, we make a cursor instance. Now this cursor instance is responsible for executing the queries. Based on the batch to made, we run the query for creating the table if the table does not exists. Here since it was batch 2017-2021. We made a batch_2017 table and then the cursor executes the query and DB connection commits to the changes.

Next we iterate over the student details stored in the different lists before and Insert the data into batch_2017 table.


+ Options

←T→	roll_no	name	branch	batch	email	ccet_email	photo_embedd
Edit Copy Delete	CO17301	Aarushi Sood	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17302	Abhijeet Baruah	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17304	Abhishek Kaushik	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17305	Aboli	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17307	Agampreet Kaur Walia	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17308	Ajay	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17309	Akhilesh Thapliyal	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17310	Amandeep	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17314	Ashish Sharma	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17315	Ashish Upadhyay	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17317	Bhuvnesh Rana	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17319	Deepanshu Garg	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]
Edit Copy Delete	CO17322	Gaurav Kaushal	cse	2017	example@domain.com	example@domain.com	[BLOB - 3.2 KiB]

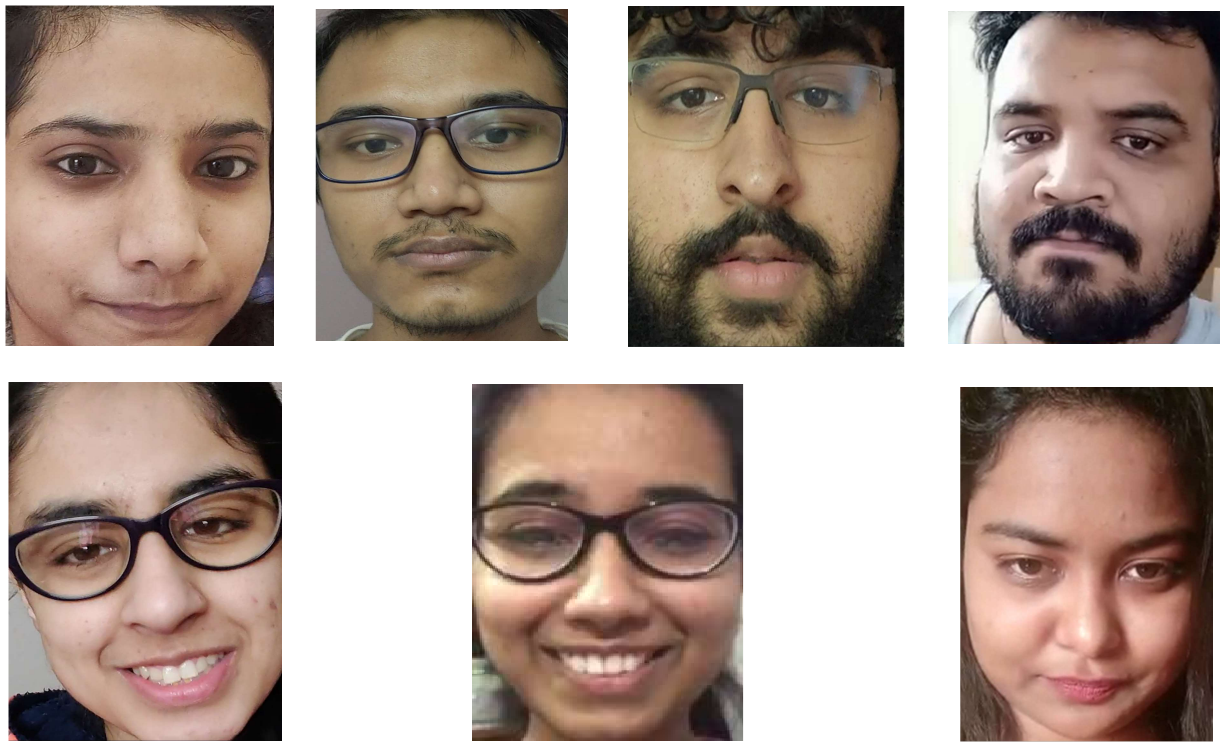
For uploading the vector of their face images, we dumps the numpy array as bytes and store it into the photo_embed column. This column is of type longblob.

This approach saves us space and time since we don’t require the Images of the students, so instead of saving the whole image, storing of embeddings help to reduce further complexity of the whole System and also avoids a repetitive step of converting an image into embeddings for face recognition.

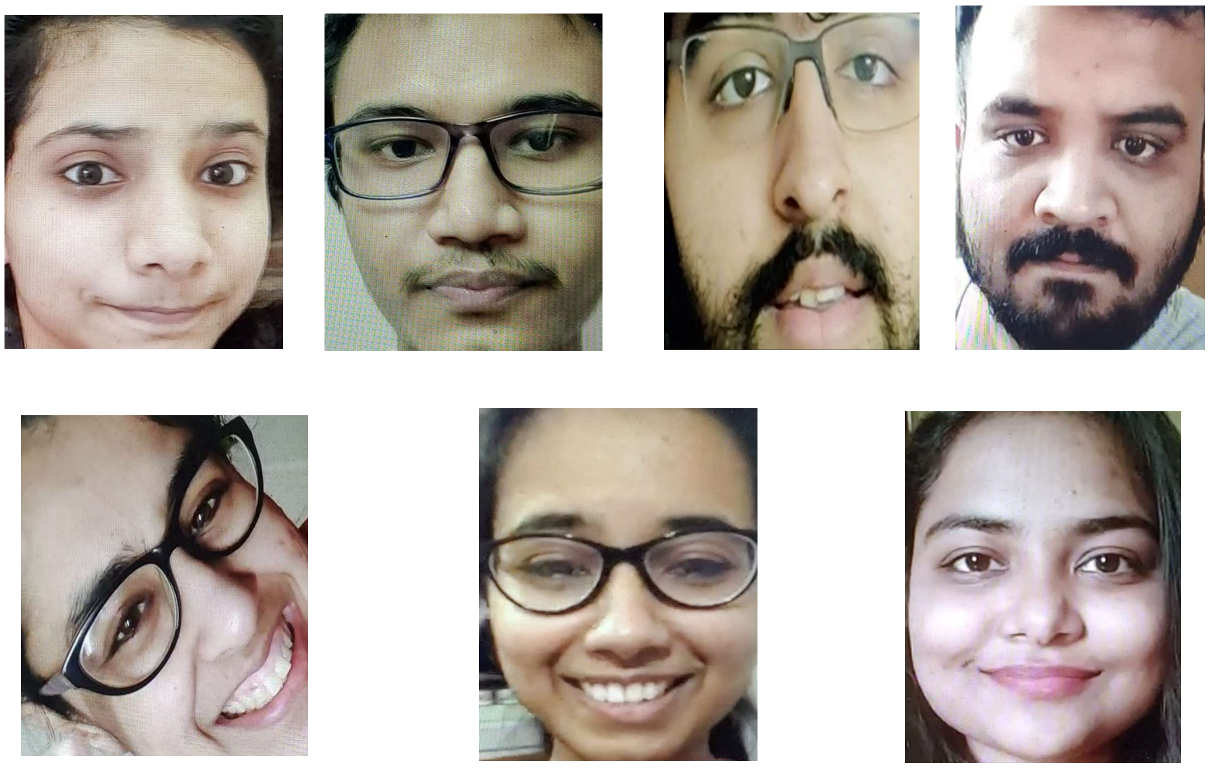
C. Training Models

Next Comes the training of the Models. So far/null, we have to train the real/ fake detector model and the mask detector model. As per tutorial on [3], we followed a similar Binary Problem approach for the development of this model. Since the shown methodology deals with the video stream in [3], we tweaked it for the photo version.

For generating the data we follow the approach of video recording us in actual presence vs in virtually present via screen. To cover the diversity we also asked our friends to volunteer in for the data, however very few agreed upon and since we had less data, the model was slightly overtrained on our faces. However, with the deployment at College level, this data collection can be ramped up and since students come from different backgrounds, the data set collected will cover different aspects of images like background light, skin tone etc. Also, the face Images were resized to 224x224x3 before feeding it into the model. Below are Some comparisons of data recorded in actual vs virtual presence.



Real Images Without Mask



Fake Faces without Mask

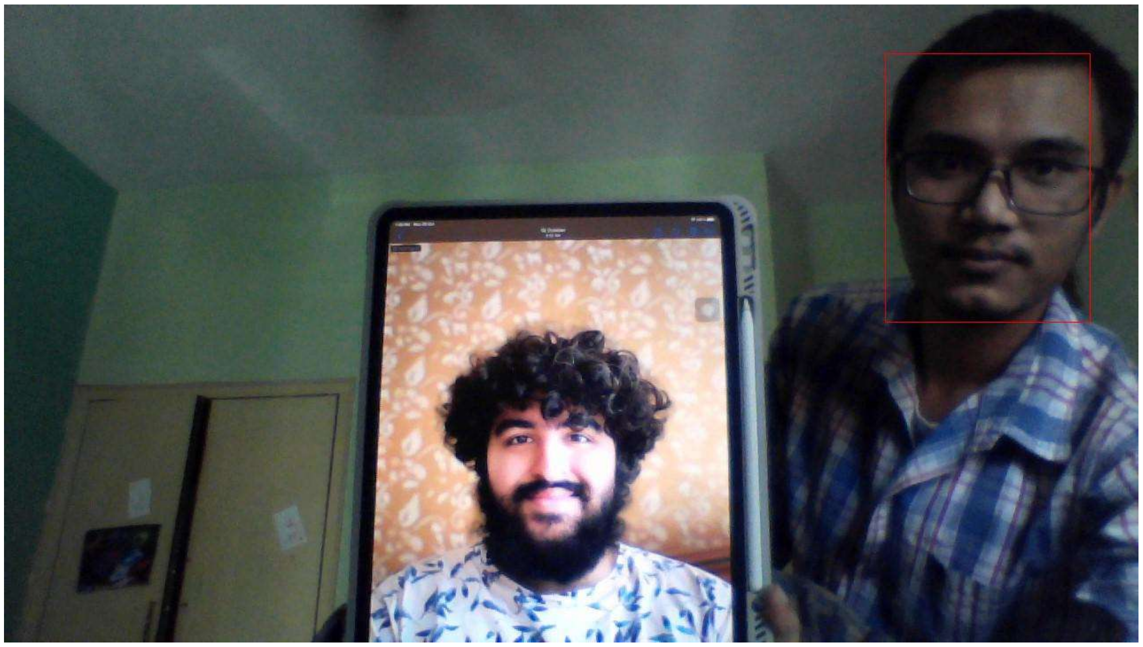
As it is visible from the above images, we have a considerable amount of difference in terms of Color accuracy, quality, detailing , pixilation, saturation, hue and brightness between the real and fake images. This can be utilised to train a Binary Classifier Deep learning CNN model that will help us to learn whether the image is real or fake and we can proceed accordingly.

Below is the code for our the real fake detector model.

```
# import the necessary packages
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras import backend as K

class RealFakeClassifier:
```

```
@staticmethod
def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last" and the channels dimension itself
        model = Sequential()
        inputShape = (height, width, depth)
        chanDim = -1

        # if we are using "channels first", update the input shape
        # and channels dimension
        if K.image_data_format() == "channels_first":
                inputShape = (depth, height, width)
                chanDim = 1

        # first CONV => RELU => CONV => RELU => POOL layer set
        model.add(Conv2D(128, (7, 7), padding="same",
                input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(Conv2D(64, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(Conv2D(32, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # second CONV => RELU => CONV => RELU => POOL layer set
        model.add(Conv2D(64, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(Conv2D(64, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(Conv2D(32, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(512))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(256))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(128))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(64))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(32))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(16))
        model.add(Activation("relu"))
        model.add(BatchNormalization())

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model
```

This is the CNN model that we tweaked to build a real/fake classifier. The model is built using tensorflow by Google.

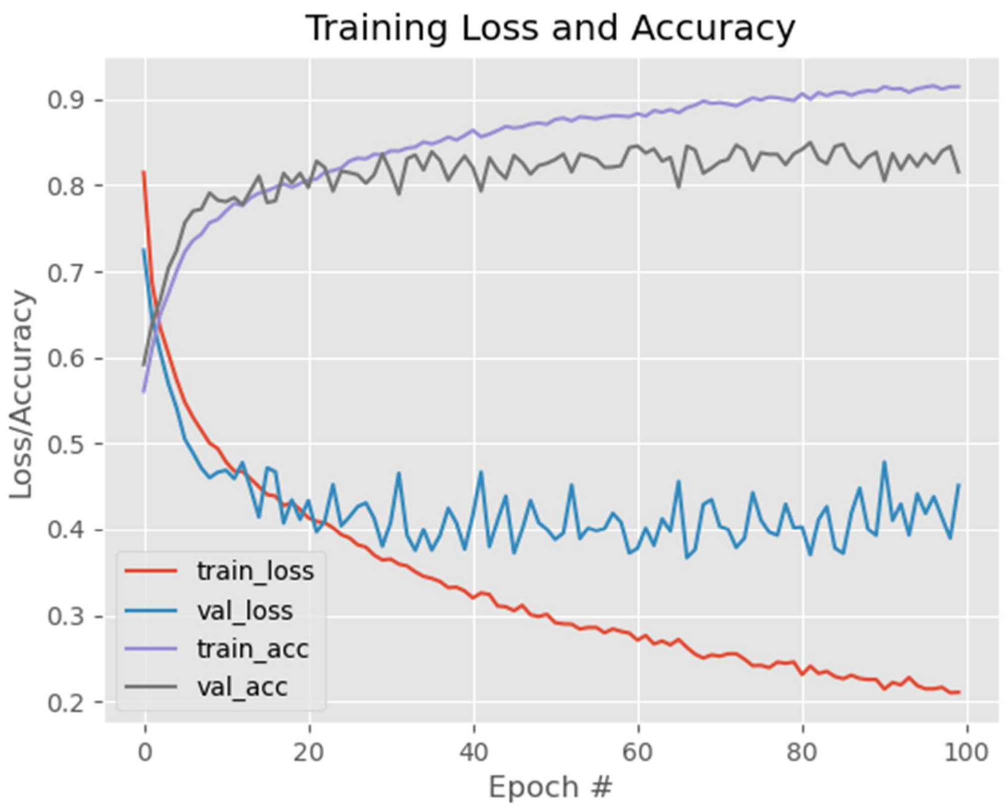


The above image shows that if a student tries to fake attendance by showing the face of his/her classmate via Laptop screen or via some tablet screen, the system's real fake model will skip over it and move to next face and thus, proxy will be avoided for such attendees.

Next we have to develop the Mask detector model for segregating faces that wore mask vs the faces that didn't. We undertook this problem too as a Binary Classifier problem. The dataset was downloaded from kaggle and consisted of 10K Images for both masked and unmasked together. The model developed for the same is in the python script shown below:

```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D, MaxPooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os

ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
        help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
        help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
        default="mask_224_detector_lite.model",
        help="path to output face mask detector model")
args = vars(ap.parse_args())

# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []
# loop over the image paths
for imagePath in imagePaths:
	# extract the class label from the filename
	label = imagePath.split(os.path.sep)[-2]
	# load the input image (224x224) and preprocess it
	image = load_img(imagePath, target_size=(224, 224))
	image = img_to_array(image)
	image = preprocess_input(image)
	# update the data and labels lists, respectively
	data.append(image)
	labels.append(label)
# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
	test_size=0.20, stratify=labels, random_state=42)
# construct the training image generator for data augmentation
aug = ImageDataGenerator(
	rotation_range=20,
	zoom_range=0.15,
	width_shift_range=0.2,
	height_shift_range=0.2,
	shear_range=0.15,
	horizontal_flip=True,
	fill_mode="nearest")

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
	input_tensor=Input(shape=(224, 224, 3)))
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dense(32, activation="relu")(headModel)
headModel = Dense(16, activation="relu")(headModel)
headModel = Dense(2, activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
        layer.trainable = False

# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
        metrics=["accuracy"])
# train the head of the network
print("[INFO] training head...")
H = model.fit(
        aug.flow(trainX, trainY, batch_size=BS),
        steps_per_epoch=len(trainX) // BS,
        validation_data=(testX, testY),
        validation_steps=len(testX) // BS,
        epochs=EPOCHS)

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
        target_names=lb.classes_))
# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save(args["model"], save_format="h5")

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

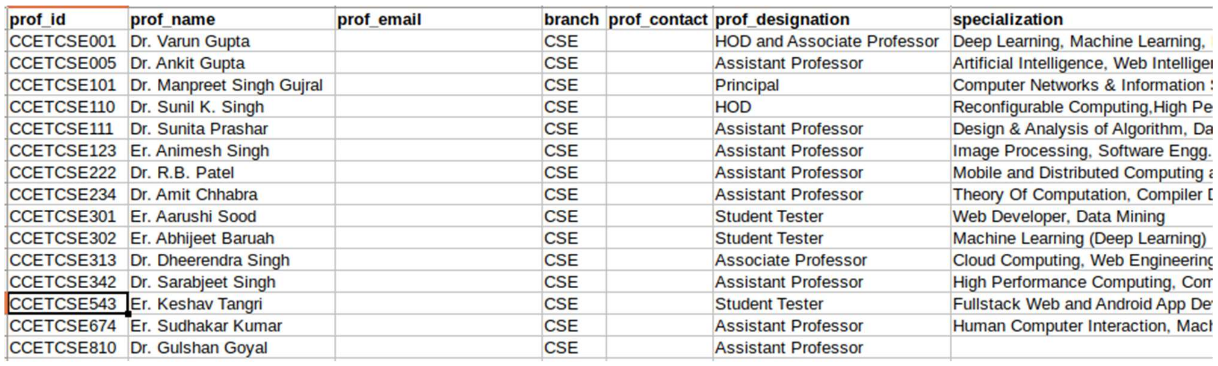
This model load some weights from the MobileNetV2 (weights from imagenet dataset) and the Fully connected Layers are tweaked per optimal settings. The model trained for 100 epochs showed the following results :



Once our models are trained, we need to structure the backend and get the things started i.e the pipeline that we discussed in models overview.

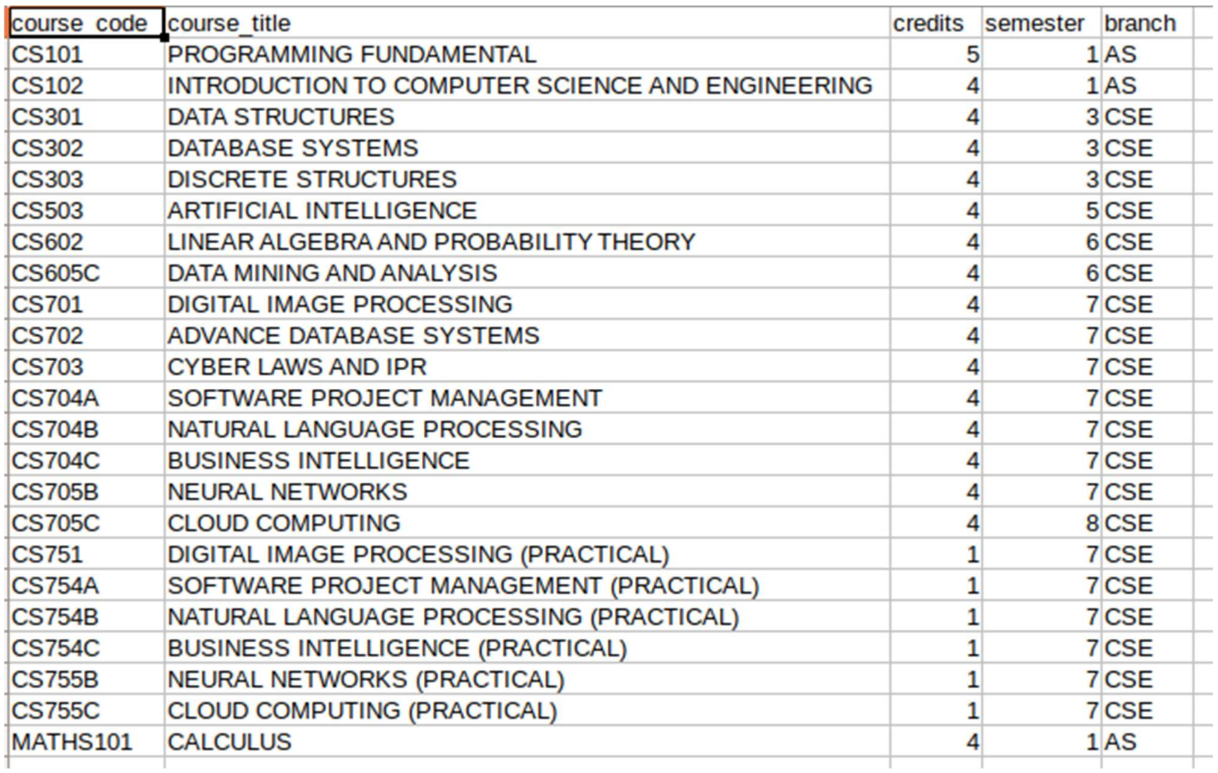
D. Struturing the Backend and setting up the APIs

For the Backend, we have a table for storing professor details. The primary ket is the teaching id of the professor.



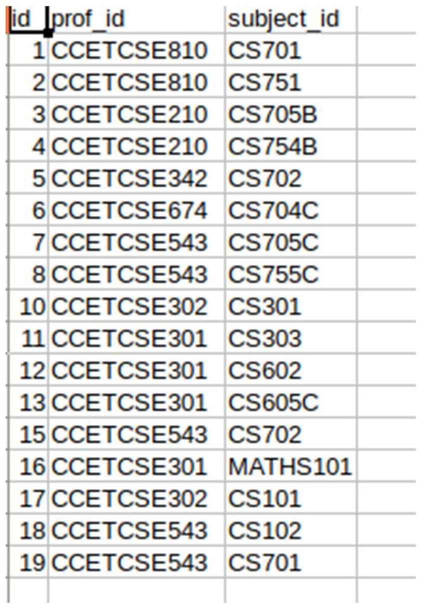
prof_id	prof_name	prof_email	branch	prof_contact	prof_designation	specialization
CCETCSE001	Dr. Varun Gupta		CSE		HOD and Associate Professor	Deep Learning, Machine Learning,
CCETCSE005	Dr. Ankit Gupta		CSE		Assistant Professor	Artificial Intelligence, Web Intellige
CCETCSE101	Dr. Manpreet Singh Gujral		CSE		Principal	Computer Networks & Information
CCETCSE110	Dr. Sunil K. Singh		CSE		HOD	Reconfigurable Computing,High Pe
CCETCSE111	Dr. Sunita Prashar		CSE		Assistant Professor	Design & Analysis of Algorithm, Da
CCETCSE123	Er. Animesh Singh		CSE		Assistant Professor	Image Processing, Software Engg.
CCETCSE222	Dr. R.B. Patel		CSE		Assistant Professor	Mobile and Distributed Computing
CCETCSE234	Dr. Amit Chhabra		CSE		Assistant Professor	Theory Of Computation, Compiler
CCETCSE301	Er. Aarushi Sood		CSE		Student Tester	Web Developer, Data Mining
CCETCSE302	Er. Abhijeet Baruah		CSE		Student Tester	Machine Learning (Deep Learning)
CCETCSE313	Dr. Dheerendra Singh		CSE		Associate Professor	Cloud Computing, Web Engineerin
CCETCSE342	Dr. Sarabjeet Singh		CSE		Assistant Professor	High Performance Computing, Com
CCETCSE543	Er. Keshav Tangri		CSE		Student Tester	Fullstack Web and Android App De
CCETCSE674	Er. Sudhakar Kumar		CSE		Assistant Professor	Human Computer Interaction, Mac
CCETCSE810	Dr. Gulshan Goyal		CSE		Assistant Professor	

Note that the email and contact details are removed to avoid leaking any private info.
These details can be added by the Admin / DB



course_code	course_title	credits	semester	branch
CS101	PROGRAMMING FUNDAMENTAL	5	1	AS
CS102	INTRODUCTION TO COMPUTER SCIENCE AND ENGINEERING	4	1	AS
CS301	DATA STRUCTURES	4	3	CSE
CS302	DATABASE SYSTEMS	4	3	CSE
CS303	DISCRETE STRUCTURES	4	3	CSE
CS503	ARTIFICIAL INTELLIGENCE	4	5	CSE
CS602	LINEAR ALGEBRA AND PROBABILITY THEORY	4	6	CSE
CS605C	DATA MINING AND ANALYSIS	4	6	CSE
CS701	DIGITAL IMAGE PROCESSING	4	7	CSE
CS702	ADVANCE DATABASE SYSTEMS	4	7	CSE
CS703	CYBER LAWS AND IPR	4	7	CSE
CS704A	SOFTWARE PROJECT MANAGEMENT	4	7	CSE
CS704B	NATURAL LANGUAGE PROCESSING	4	7	CSE
CS704C	BUSINESS INTELLIGENCE	4	7	CSE
CS705B	NEURAL NETWORKS	4	7	CSE
CS705C	CLOUD COMPUTING	4	8	CSE
CS751	DIGITAL IMAGE PROCESSING (PRACTICAL)	1	7	CSE
CS754A	SOFTWARE PROJECT MANAGEMENT (PRACTICAL)	1	7	CSE
CS754B	NATURAL LANGUAGE PROCESSING (PRACTICAL)	1	7	CSE
CS754C	BUSINESS INTELLIGENCE (PRACTICAL)	1	7	CSE
CS755B	NEURAL NETWORKS (PRACTICAL)	1	7	CSE
CS755C	CLOUD COMPUTING (PRACTICAL)	1	7	CSE
MATHS101	CALCULUS	4	1	AS

This is the subjects table storing the course code, title, credits, semester to be taught to and branch. In short, all the detail about the subject is stored in this table. Next we have to deal with the mapping of the teachers to subjects. This is done by maintaining another table.



id	prof_id	subject_id
1	CCETCSE810	CS701
2	CCETCSE810	CS751
3	CCETCSE210	CS705B
4	CCETCSE210	CS754B
5	CCETCSE342	CS702
6	CCETCSE674	CS704C
7	CCETCSE543	CS705C
8	CCETCSE543	CS755C
10	CCETCSE302	CS301
11	CCETCSE301	CS303
12	CCETCSE301	CS602
13	CCETCSE301	CS605C
15	CCETCSE543	CS702
16	CCETCSE301	MATHS101
17	CCETCSE302	CS101
18	CCETCSE543	CS102
19	CCETCSE543	CS701

This is the professor_subjects table that stores the mapping of a professor to a subject.
This kind of segregation allows a subject to be taught by same teacher if the subject is to be taught in different branch. It also allows students to choose teachers for a particular subject.

The next challenge that comes up is how to store the attendance for a particular subject taught by a particular teacher to a particular branch of a particular batch? Above that it should work flawlessly dynamically without requiring to restructure the things.

For this we brainstormed over the approach that, based on the fact that a subject is taught in a particular semester, we can fetch the batch to which it is being taught.

Say a subject is of 8th semester and current going session is 2021, so the batch that is in 8th semester in 2021 will be = 2021 – (8/2) = Batch_2017.

For odd semesters, the batch that will be in 7th semester in 2021 (that means july to dec), will be 2021 – (7/2) = Batch_2018. We take integer division.

This shows that from subject semester, we can get the batch of the student. Next from the subject branch , we can get which branch it is being taught.

The uniqueness of subject can be understood from its course code. And to map this to a unique teacher we can use prof id.

Therefore we came down to an approach of making a table if not exists with the name

profid_courseid_branch_batch

Eg. CCETCSE810_CS701_CSE_2017 for teacher Dr. Gulshan Goyal, teaching batch 2017 of CSE Branch the subject Digital Image Processing CS701.
Another Important thing is that when the table is made for the first time, the first column is automatically filled with the roll numbers of the students of batch_2017, CSE and the attendance

will be marked in the following available column which is created with the column name as the date of attendance.

E. Combing All

Now we have described our backend structure, developed Deep learning models, understood our requirements, we require to combine the approach and develop a script that will import the models, take input from the frontend or some another API, take in Image to be processed, approach the image as per models overview and then as per point D, update attendance on the Database. The script for this purpose is the vectorized.py script that is shown below. In our approach with vectors, we have utilised vectorization to parallelly process the Image recognition instead of iteration and We have saved considerable amount of time with this approach.

```
#Import Packages
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # or any {'0', '1', '2'}
from tensorflow.keras.preprocessing.image import img_to_array,array_to_img
from tensorflow.keras.models import load_model
from keras_facenet import FaceNet
#from mtcnn.mtcnn import MTCNN
import mysql.connector
from PIL import Image
import pandas as pd
import numpy as np
import argparse
import warnings
import pickle
#import cv2

warnings.filterwarnings("ignore")
#Global List for storing presenties roll number
attendence = []

#Argument Parser
ap = argparse.ArgumentParser()
#Image arg parser
ap.add_argument("-i","--image",required = True,
                                    help = 'path to input images')

#Self trained Mask Detector model
ap.add_argument("-m","--model",type = str,
                                    default = 'mask_224_detector.model',
                                    help='path to trained mask detector model')

#Self trained Real Fake Model
ap.add_argument('-rf','--rf_model',type = str,
                                    default = 'model_wo_mask/real_fake_wo_mask.model',
                                        help = 'path to trained real fake model')

#Real Fake model labels
ap.add_argument('-rfl','--rf_labels',type = str,
                                        default = 'model_wo_mask/real_fake_wo_mask_labels.pickle',
                                        help = 'path to labels for real fake')

#Self trained Real Fake Model With Mask
ap.add_argument('-rfm','--rfm_model',type = str,
                                        default = 'model_w_mask/real_fake_w_mask.model',
                                        help = 'path to trained real fake model with mask')

#Real Fake model labels with mask
ap.add_argument('-rflm','--rfm_labels',type = str,
                                        default = 'model_w_mask/real_fake_w_mask_labels.pickle',
                                        help = 'path to labels for real fake model with mask')

ap.add_argument("-c", "--confidence", type=float, default=0.5,
          help="minimum probability to filter weak detections")

args = vars(ap.parse_args())


#classroom path
class_path = "classroom"
#Final Image path
face_path = "images"

def fetchresult(branch,batch):
    table_name = 'batch_'+str(batch)
    conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="eyeattend"
    )
    if conn.is_connected():
        print("Successfully connected")
        df = pd.read_sql(f"SELECT roll_no,photo_embedd FROM {table_name} WHERE
branch=(%s) AND batch = (%s)",con=conn,params=(branch,batch))
        #results = mycursor.fetchall()
        #df = pd.DataFrame(results)
        conn.commit()
        print('success in fetching')
        # Closing the connection
        conn.close()
        return df

    else:
        print('Error in connecting with database')
        return -1



def calculate_attendance(image):
    #######here using the model we will convert the image to embeddings
    # Gets a detection dict for each face
    # in an image. Each one has the bounding box and
    # face landmarks (from mtcnn.MTCNN) along with
    # the embedding from FaceNet.
    imgarray = img_to_array(image)
    detections = embedder.extract(imgarray, threshold=0.95)


    length = len(detections)
    for i in range(length):
        x1, y1, width, height = detections[i]['box']
        #calculating Box Dims
        x1, y1 = abs(x1), abs(y1)
        x2, y2 = x1 + width, y1 + height

        #Taking face from Complete image
        face_im = imgarray[y1:y2, x1:x2]
        face = array_to_img(face_im)
        face = img_to_array(face.resize((224,224)))
        face = np.expand_dims(face, axis=0)
                #detecting for mask
        (mask,withoutMask) = mask_detector.predict(face)[0]

        if mask > withoutMask:
            print("Mask")
            #Mask conditions

        else:
            print("No Mask")
            #No mask Conditions
                        #Will check for real and fake now
            label = rf_le.classes_[np.argmax(real_fake_detector.predict(face)[0])]
            #Checking label
            if label == "real":
                            #We will do stuff here and ignore for fake faces
                            #NO PROXY ALLOWED !!!
            embedding = embedder.embeddings(face)

            difference = picklefiles-embedding
            normedvector = np.sum(np.abs(difference)**2,axis=-1)**(1./2)
            index=np.argmin(normedvector)
            if normedvector[index] < 0.9:
                attendence.append(results['roll_no'][index])
                print(results['roll_no'][index],normedvector[index])
            else:
                print("Not in the Database")

    return "JOB DONE !"


results = fetchresult("cse","2017")
if isinstance(results, int):
    if results == -1:
        exit(0)
picklefiles = [np.array(pickle.loads(results['photo_embedd'][i])) for i in range(len(results))]
picklefiles=np.squeeze(picklefiles)


#Loading all Models
print("Loading All Models...")
mask_detector = load_model(args['model'])
print("Mask Detector loaded !")
embedder = FaceNet()
#face_detector = MTCNN()
print("Facenet_MTCNN Loaded")
#model_face = load_model('facenet_keras.h5')
print("Facenet Loaded")
real_fake_detector = load_model(args['rf_model'])
print("Real fake without mask loaded")


#Reading real fake labels
rf_le = pickle.loads(open(args["rf_labels"], "rb").read())
#rf_le_w_mask = pickle.loads(open(args["rfm_labels"], "rb").read())
print("All Models Loaded")



#reading Image
image = Image.open(args['image'])
#Detecting Faces
calculate_attendance(image)
```

```
with open('attendence.txt', 'w') as f:
    for item in attendence:
        f.write("%s\n" % item)

#image = array_to_img(pixels)
#image.save(face_path + "/final.jpg")
```

Explanation : The code begins by downloading the student details from the MySQL Database via the fetchresult() function that accepts branch, batch and table name as parameters. This function also calls checkTableeExists() function, that will create the final data storage table in the format **profid_courseid_branch_batch if it does not exists.** The photo_embed column is converted back to numpy array with the resulting dimensions of 45 x 1 x 512. Then with the numpy.squeeze function, we change it to 45 x 512 for easy calculations. Next all the developed models are imported along with their labels. Further, with the insertion of the image path, calculate attendance is called that accepts image parameter. The image is converted to array and using the pretrained keras_facenet model with a threshold of 95%, we detect faces in the particular image. Next, we iterate over the respective faces, store the coordinates of the bounding box, extract the Region of Interest and convert it into array. Resize it to 224x224x3 and then run over the Mask detector model, that returns either mask or no mask, since we don't have mask data, we just work for the condition where mask is false, then we run the face image over real fake detector, if it is fake, it is not worked upon else, vectorization with the all class images (45x512) is performed. This is where the Facial recognition is going on and this will return 3the Roll Number of the student if the student is of this class. We store this in a list data structure. Once we are done iterating over the whole collection of detected faces, we connect with the Database and first, check if the column with current day's date exists else we alter the table and add the column to it. Next using pandas we upload the attendance over the database. This data is then sent to frontend app so that the data can also be shown to the teacher using the app.

IV. BUILDING THE FRONTEND

The explanation of the development of front end is beyond the scope of this report, we will demonstrate the flow of the app with screenshots. The complete working of Frontend app with explanation is shown in the video link:
https://drive.google.com/file/d/1BO7evOKZDRKVQ1dijzrxLdv12u3DMaIA/view?usp=sharing

REFERENCES

[1]. https://www.electronicid.eu/en/blog/post/face-recognition/en
[2]. https://github.com/faustomorales/keras-facenet
[3]. https://www.pyimagesearch.com/2019/03/11/liveness-detection-with-opencv/