**Name:** M. Abdul Moeez Atlas

**Reg No:** FA17-BCS-040

**Class:** BCS-6A

**Submitted to:** Mam Shahela Saif

# Computer Vision Final Project Report

**Project Name:**

Face Recognition Attendance System

**Working Principle:**

1. Encode a picture using **the HOG (Histogram of Gradients) algorithm** to create a simplified version of the image. Using this simplified image, find the part of the image that most looks like a generic HOG encoding of a face.
2. Figure out the pose of the face by finding the main landmarks in the face. Once we find those landmarks, use them to warp the image so that the eyes and mouth are centered. This is done by **face landmark estimation algorithm**.
3. Pass the centered face image through a **pre-trained deep convolutional neural network** that knows how to measure features of the face. Save those 128 measurements.
4. Looking at all the faces we have measured in the past, see which person has the closest measurements to our face’s measurements. That’s our match. This is done through **SVM (Support Vector Machine) classifier** but lots of classification algorithms could work.

**Software and Libraries Used:**

* Pycharm
* cmake
* dlib
* face\_recognition (<https://github.com/ageitgey/face_recognition>)
* numpy
* opencv-python

**Code :**

**// Imports of relevant libraries.**

import cv2

import numpy as np

import face\_recognition

import os

from datetime import datetime

**// Generating student names and images array from the provided path**

path = 'Students'

images = []

classNames = []

myList = os.listdir(path)

print(myList)

for cl in myList:

    curImg = cv2.imread(f'{path}/{cl}')

    images.append(curImg)

    classNames.append(os.path.splitext(cl)[0])

print(classNames)

**// Function for loading Images and converting to RGB. Then we encode the image using HOG algorithm and afterwards also generate the 128 special face measurements unique to every face and store in our array .**

def findEncodings(images):

    encodeList = []

    for img in images:

        img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

        encode = face\_recognition.face\_encodings(img)[0]

        encodeList.append(encode)

    return encodeList

**// Attendance generating function**

def markAttendance(name):

    with open('Attendance.csv', 'r+') as f:

        myDataList = f.readlines()

        nameList = []

        for line in myDataList:

            entry = line.split(',')

            nameList.append(entry[0])

        if name not in nameList:

            now = datetime.now()

            dtString = now.strftime('%H:%M:%S')

            f.writelines(f'\n{name},{dtString}')

**// Encoding function call. Afterwards we find faces locations and encodings.**

**Compare\_faces function** is used to match the faces. Similarly, **face\_distance function** is used to find how likely the faces match in terms of numbers. This is helpful when more than one faces are in images.

We will get the name of the student displayed on screen in case of a match and unknown being displayed in case of no match. Attendance is marked in the Csv file during runtime.

encodeListKnown = findEncodings(images)

print('Encoding Complete')

cap = cv2.VideoCapture(0)

while True:

    success, img = cap.read()

    # img = captureScreen()

    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)

    imgS = cv2.cvtColor(imgS, cv2.COLOR\_BGR2RGB)

    facesCurFrame = face\_recognition.face\_locations(imgS)

    encodesCurFrame = face\_recognition.face\_encodings(imgS, facesCurFrame)

    for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):

        matches = face\_recognition.compare\_faces(encodeListKnown, encodeFace)

        faceDis = face\_recognition.face\_distance(encodeListKnown, encodeFace)

        # print(faceDis)

        matchIndex = np.argmin(faceDis)

        if faceDis[matchIndex] < 0.50:

            name = classNames[matchIndex].upper()

            markAttendance(name)

        else:

            name = 'Unknown'

        # print(name)

        y1, x2, y2, x1 = faceLoc

        y1, x2, y2, x1 = y1 \* 4, x2 \* 4, y2 \* 4, x1 \* 4

        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

        cv2.rectangle(img, (x1, y2 - 35), (x2, y2), (0, 255, 0), cv2.FILLED)

        cv2.putText(img, name, (x1 + 6, y2 - 6), cv2.FONT\_HERSHEY\_COMPLEX, 1, (255, 255, 255), 2)

    cv2.imshow('Webcam', img)

    cv2.waitKey(1)

**Working Snapshot:**

**A screenshot of a television screen

Description automatically generated**

**Attendance Report Example:**

**A screenshot of a computer

Description automatically generated**