FACIAL RECOGNITION & ATTENDANCE SYSTEM IN MACHINE LEARNING

MINI PROJECT REPORT

Submitted by

KISHORE P	211718104067
PONMOZHI R	211718104099
PRADAKSHINA L H	211718104100

In partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report **“FACIAL RECOGNITION & ATTENDANCE SYSTEM USING MACHINE LEARNING”**, is the Bonafide work of **KISHORE** P (**211718104067**) **PRADAKSHINA** L H (**211718104100**), **and PONMOZHI R (211718104099)** who carried out the mini project work under my supervision.

SIGNATURE:

Dr. D.C.JOY WINNIE WISE, B.E., M.E., PhD,

HEAD OF THE DEPARTMENT,

Professor,

Dept. of Computer Science,

and Engineering.

Rajalakshmi Institute of,
Technology,
Kuthambakkam Post,

Chennai - 600 124

SIGNATURE:

Dr. R. LALITHA, M.E., PhD,

SUPERVISOR,

Professor,

Dept. of Computer Science

and Engineering.

Rajalakshmi Institute of
Technology,
Kuthambakkam

Post, Chennai- 600 124

ACKNOWLEDGEMENT

We express our sincere gratitude to our honorable Chairperson Dr.(Mrs.) THANGAM MEGANATHAN, M.A., M.Phil., Ph.D., **and** Chairman **Thiru. S. MEGANATHAN, B.E., F.I.E.,** for their constant encouragement to do this mini project and also during the entire course period.

We thank our Principal **Dr. P. K. NAGARAJAN** and our Head of the department **Dr. D.C. JOY WINNIE WISE, B.E., M.E., Ph.D,** for their valuable suggestions and guidance for the development and completion of this mini project.

Words fail to express our gratitude to our Mini Project coordinators **L. MARIA MICHAEL VISUVASAM, M.E., M.B.A., Ph.D., Dr. O. PANDITHURAI, M.E., Ph.D., Mr. K. DHINAKARAN, M.E.,** and

Internal guide **Dr. R. LALITHA, M.E., Ph.D.,** who took special interest in our mini project and gave their consistent support and guidance during all stages of this mini project.

Finally, we thank all the teaching and non-teaching faculty members of the **DEPARTMENT OF COMPUTER SCIENCE AND**

ENGINEERING of our college who helped us to complete this mini project.

Above all we thank our parents and family members for their constant support and encouragement for completing this mini project.

TABLE OF CONTENT

ABSTRACT

This system is developed for deploying an easy and a secure way of taking down attendance. The program first stores the images of all the authorized persons and store the information in database. The system stores the image by mapping it into a coordinate structure. Next time whenever the registered person comes in for attendance the camera recognizes the person and marks his attendance along with time.

Identifying a person with an image has been popularized through the mass media. However, it is less robust to fingerprint or retina scanning. This report describes the face detection and recognition mini-project undertaken for the visual perception and autonomy module at Plymouth university. It reports the technologies available in the Open-Computer-Vision (OpenCV) library and methodology to implement them using Python. For face detection, Haar-Cascades were used and for face recognition Eigenfaces, Fisher faces and Local binary pattern histograms were used. The methodology is described including flow charts for each stage of the system. Next, the results are shown including plots and screen-shots followed by a discussion of encountered challenges. The report is concluded with the authors' opinion on the project and possible applications.

INTRODUCTION

The following document is a report on the mini project for Robotic visual perception and autonomy. It involved building a system for face detection and face recognition using several classifiers available in the open computer vision library (OpenCV). Face recognition is a non-invasive identification system and faster than other systems since multiple faces can be analyzed at the same time. The difference between face detection and identification is, face detection is to identify a face from an image and locate the face. Face recognition is making the decision" whose face, is it? ", using an image database. In this project both are accomplished using different techniques and are described below. The report begins with a brief history of face recognition. This is followed by the explanation of HAAR-cascades, Eigenface, Fisher face and Local binary pattern histogram (LBPH) algorithms. Next, the methodology and the results of the project are described. A discussion regarding the challenges and the resolutions are described. Finally, a conclusion is provided on the pros and cons of each algorithm and possible implementations.

LITERATURE SURVEY

1.www.medium.com – Machine learning is Fun! Part 4: Modern Face Recognition with Deep Learning / by ADAM GEITGEY

2.www.computervision.com – Author-Murtaza Hassan

3.www.PyCharm.com – Software provider for OpenCV

4.www.python.com – Python (3.9.5)

5. https://www.researchgate.net/publication/318900718

6.https://www.researchgate.net/publication/318900718_Face_Detection_Face_ Recognition_Using_Open_Computer_Vision_Classifies

7. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

SYSTEM STUDY

FEASABILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.
Three key considerations involved in the
feasibility analysis are
** ECONOMICAL FEASIBILITY*
**TECHNICAL FEASABILITY*
**SOCIAL FEASABILITY*

ECONOMICAL FEASABILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASABILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASABILITY

The aspect of study is to check the level of acceptance of the system by the user. This acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system

SYSTEM REQUIREMENTS:

SOFTWARE REQUIREMENTS:

Operating System	*:*	*WINDOWS*
Simulation Tool	*:*	*OPENCV*
Documentation	*:*	*MS—OFFICE*
Coding platform	*:*	*PyCharm*
Domain	*:*	*Python*

HARDWARE REQUIREMENTS:

CPU type	*: Intel Pentium 4*
Clock speed	*:3.0 GHz*
Ram size	*:512 MB*
Hard disk capacity	*:80 GB*
Monitor type	*:15 Inch color monitor*
Keyboard type	*: Internet keyboard*
CD -drive type	*: 52xmax*

THE HISTORY OF FACE RECOGNITION

Face recognition began as early as 1977 with the first automated system being introduced By Kanade using a feature vector of human faces [1]. In 1983, Sirovich and Kirby introduced the principal component analysis (PCA) for feature extraction [2]. Using PCA, Turk and Pentland Eigenface were developed in 1991 and is considered a major milestone in technology [3]. Local binary pattern analysis for texture recognition was introduced in 1994 and is improved upon for facial recognition later by incorporating Histograms (LBPH) [4], [5]. In 1996 Fisher face was developed using Linear discriminant analysis (LDA) for dimensional reduction and can identify faces in different illumination conditions, which was an issue in Eigenface method [6]. Viola and Jones introduced a face detection technique using HAAR cascades and ADAB oost [7]. In 2007, A face recognition technique was developed by Naruniec and Skarbek using Gabor Jets that are similar to mammalian eyes [8], [9]. In This project, HAAR cascades are used for face detection and Eigenface, Fisher face and LBPH are used for face recognition.

How to use Machine Learning on a Very Complicated Problem

face recognition is really a series of several related problems:

1. *First, look at a picture and find all the faces in it*
2. *Second, focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.*
3. *Third, be able to pick out unique features of the face that you can use to tell it apart from other people— like how big the eyes are, how long the face is, etc.*
4. *Finally, compare the unique features of that face to all the people you already know to determine the person's name.*

As a human, your brain is wired to do all of this automatically and instantly. In fact, humans are too good at recognizing faces and end up seeing faces in everyday objects:

Computers are not capable of this kind of high-level generalization (at least not yet…), so we have to teach them how to do each step in this process separately.

We need to build a pipeline where we solve each step of face recognition separately and pass the result of the current step to the next step. In other words, we will chain together several machine learning algorithms:

1.find face in the image
2.Analyze Facial Features
3.Compare Against Know Faces
4.Make a Prediction

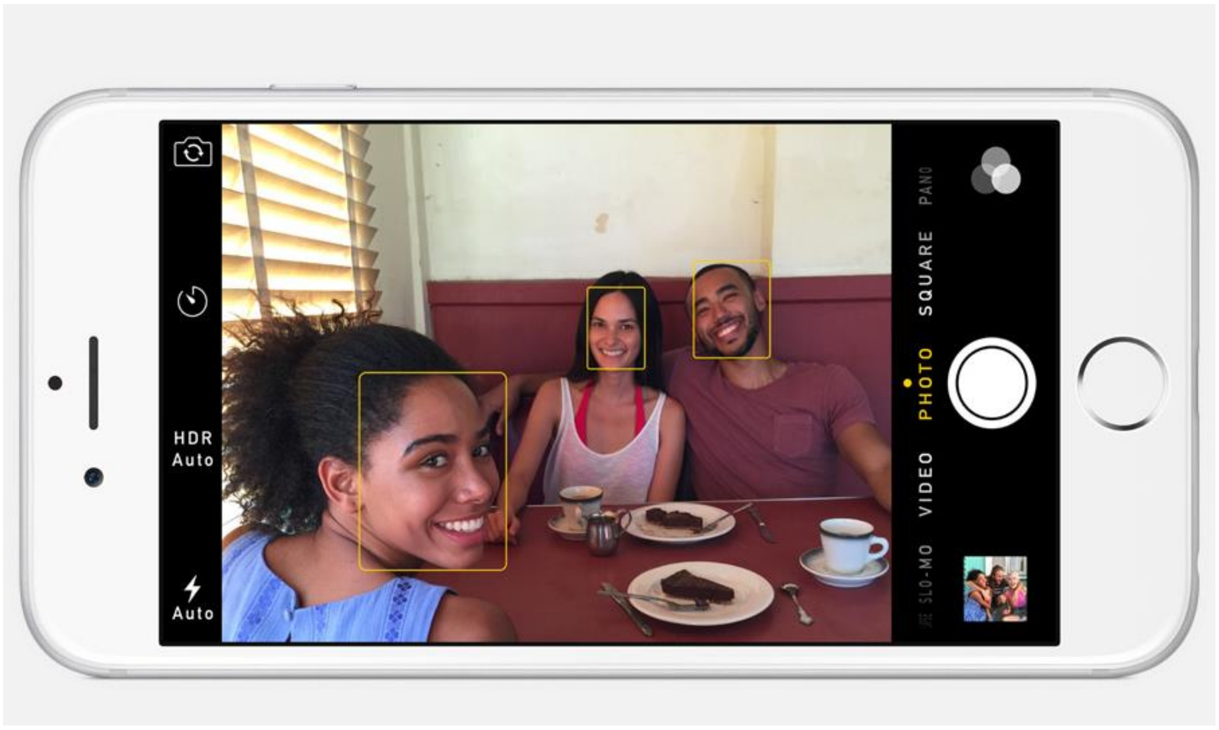
FACE RECOGNITION - STEP BY STEP

Let's tackle this problem one step at a time. For each step, we'll learn about a different machine learning algorithm. I'm not going to explain every single algorithm completely to keep this from turning into a book, but you'll learn the main ideas behind each one and you'll learn how you can build your own facial recognition system in Python using OpenFace and dlib.

Step 1: finding all the faces

The first step in our pipeline is face detection. Obviously, we need to locate the faces in a photograph before we can try to tell them apart!

If you've used any camera in the last 10 years, you've probably seen face detection in action:



Face detection is a great feature for cameras. When the camera can automatically pick out faces, it can make sure that all the faces are in focus before it takes the picture. But we'll use it for a different purpose — finding the areas of the image we want to pass on to the next step in our pipeline.

Face detection went mainstream in the early 2000's when Paul Viola and Michael Jones invented a way to detect faces that was fast enough to run on cheap cameras. However, much more reliable solutions exist now. We're going to use a method invented in 2005 called Histogram of Oriented Gradients — or just **HOG** *for short.*

To find faces in an image, we'll start by making our image black and white because we don't need color data to find faces:

Then we'll look at every single pixel in our image one at a time. For every single pixel, we want to look at the pixels that directly surrounding it

Our goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then we want to draw an arrow showing in which direction the image is getting darker

If you repeat that process for ***every single pixel*** *in the image, you end up with every pixel being replaced by an arrow. These arrows are called gradients and they show the flow from light to dark across the entire image*

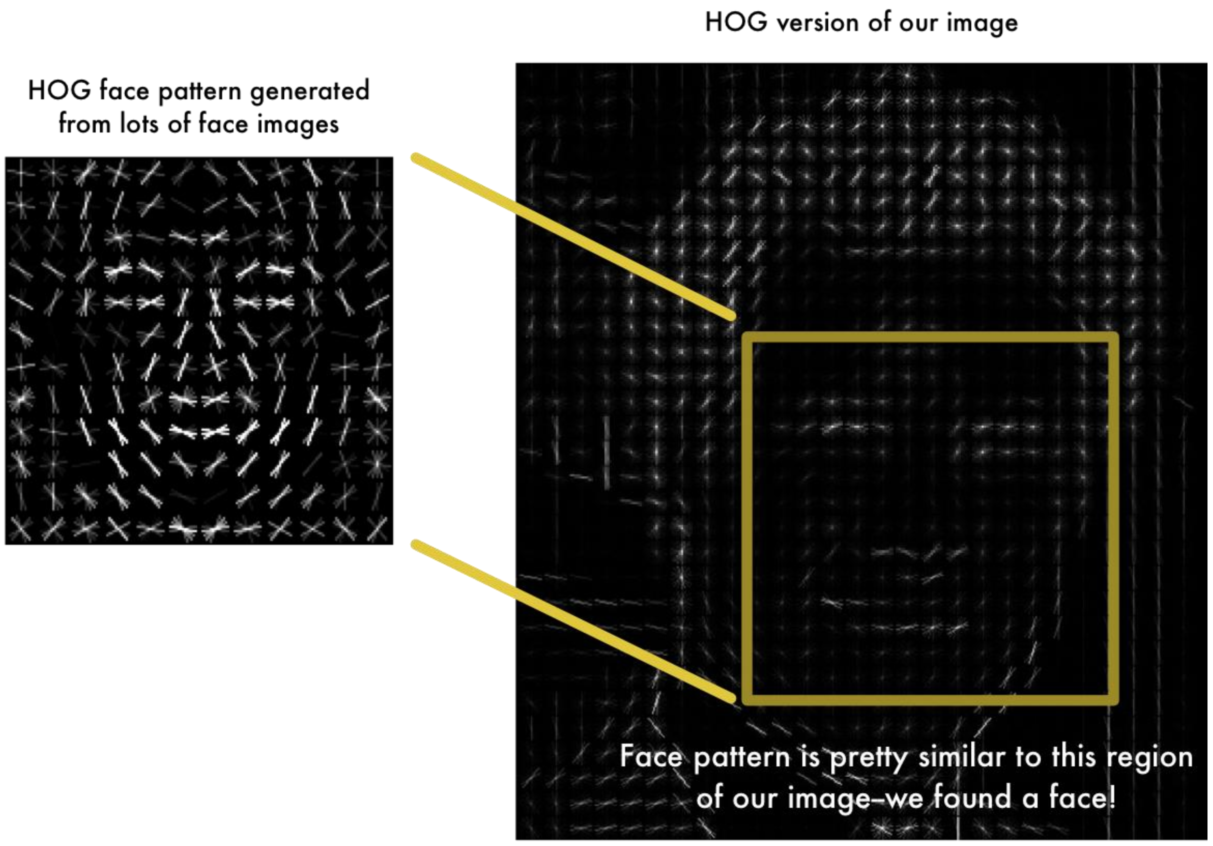
This might seem like a random thing to do, but there's a really good reason for replacing the pixels with gradients. If we analyze pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the direction *that brightness changes, both really dark images and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve!*

But saving the gradient for every single pixel gives us way too much detail. We end up missing the forest for the trees. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

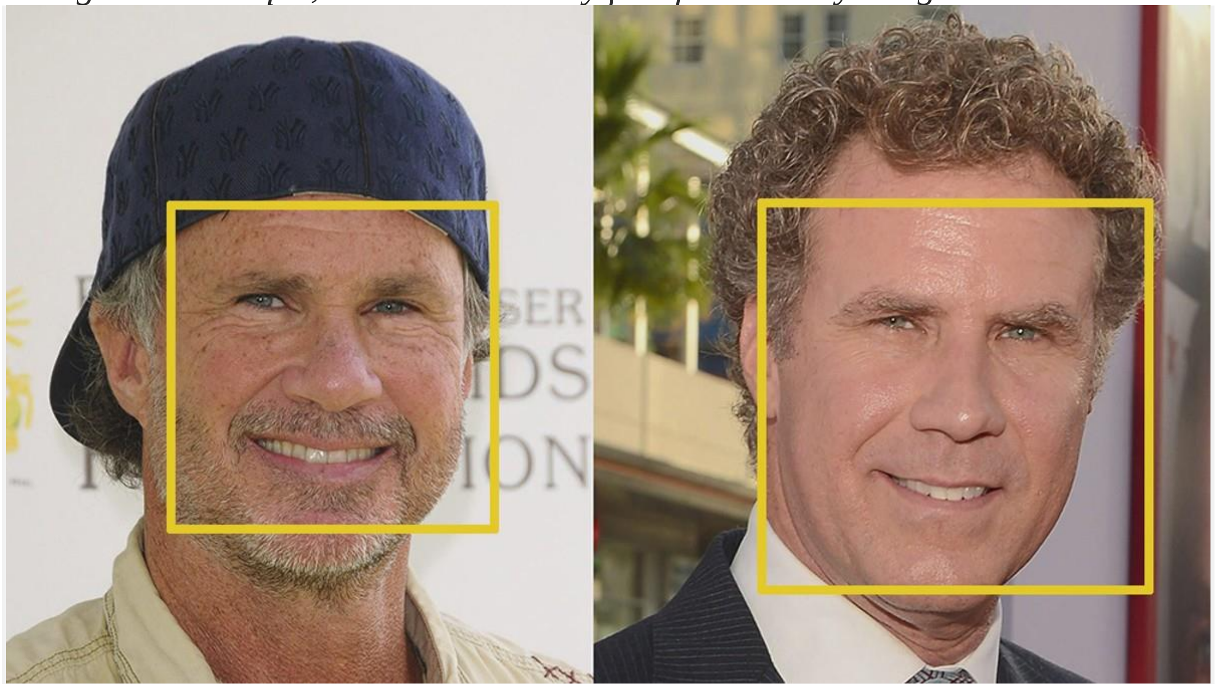
To do this, we'll break up the image into small squares of 16x16 pixels each. In each square, we'll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc….). Then we'll replace that square in the image with the arrow directions that were the strongest.

The end result is we turn the original image into a very simple representation that captures the basic structure of a face in a simple way

To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces:

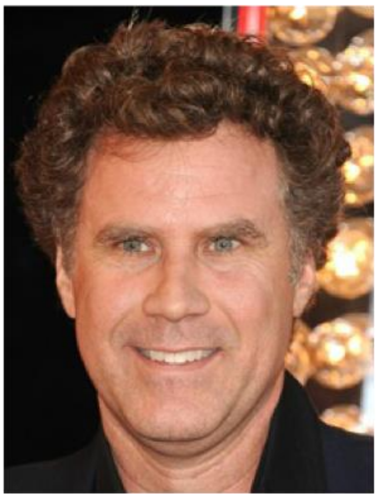
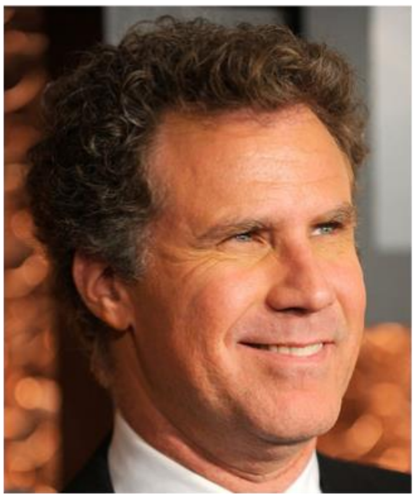

HOG version of our image
HOG face pattern generated from lots of face images
Face pattern is pretty similar to this region of our image–we found a face!

Using this technique, we can now easily find faces in any image:



Step 2: posing and projecting Faces

Whew, we isolated the faces in our image. But now we have to deal with the problem that faces turned different directions look totally different to a computer:

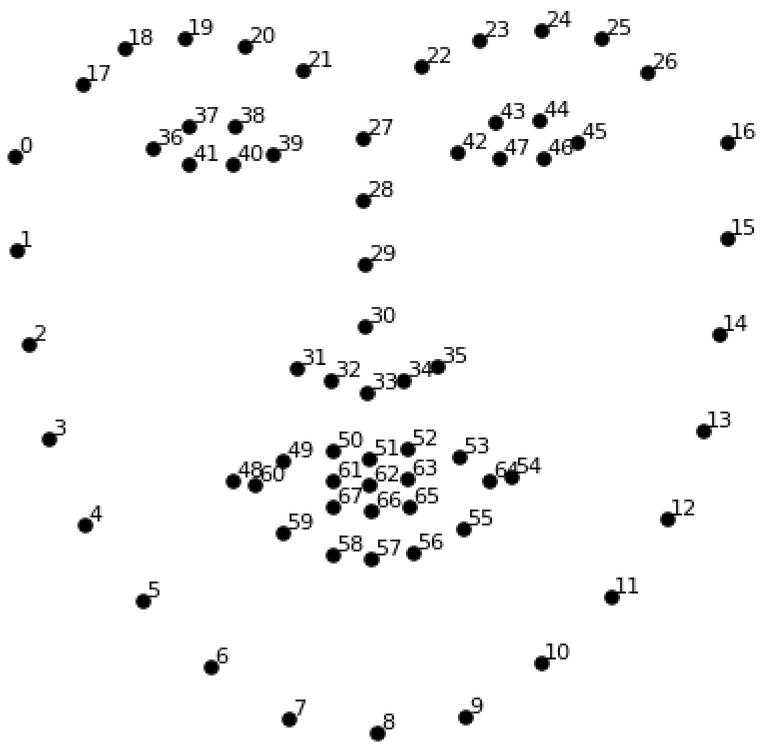



humans can easily recognize that both images are of Will Ferrell, but computers would see these pictures as two completely different people.

To account for this, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare faces in the next steps.

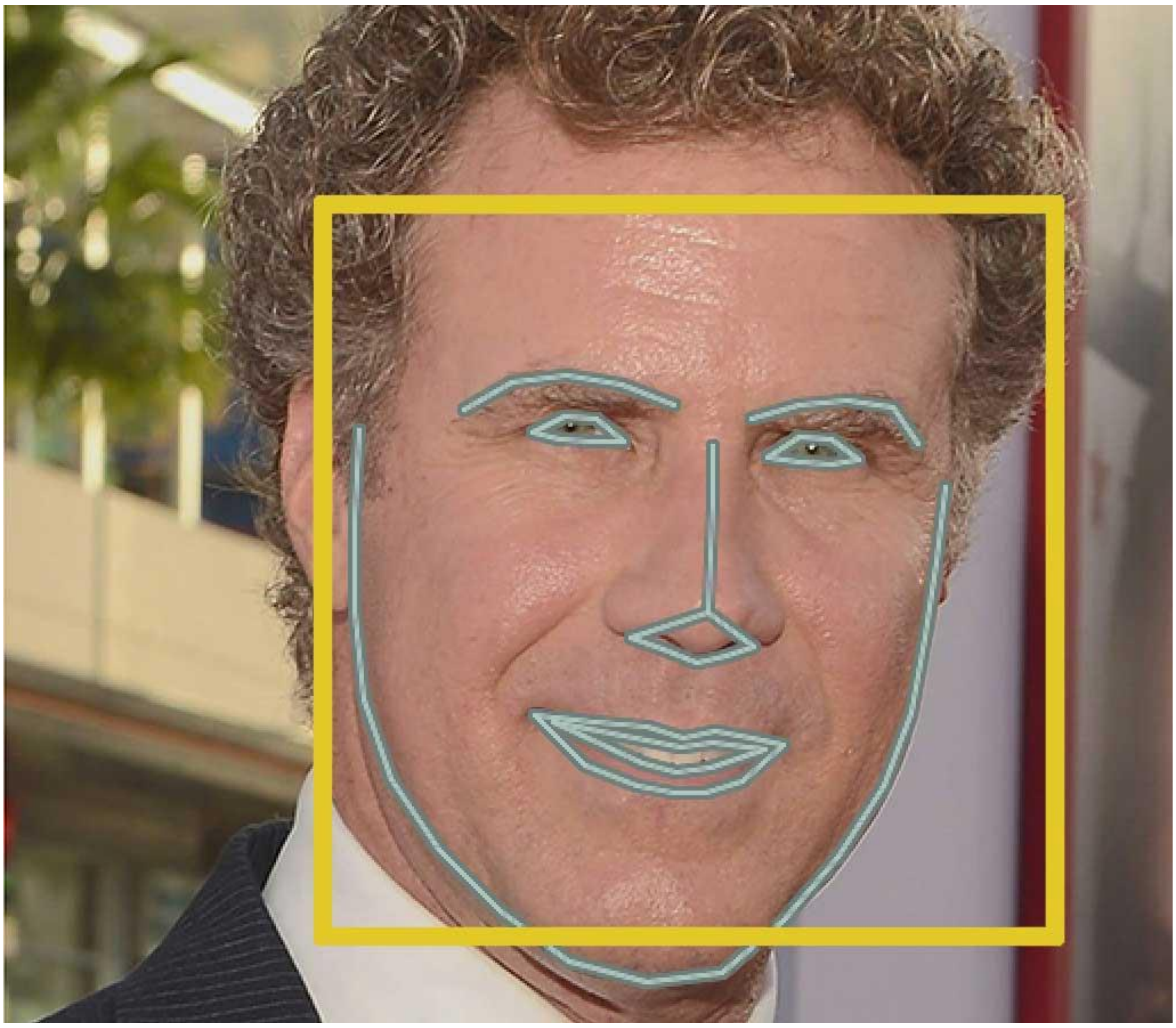
To do this, we are going to use an algorithm called **face landmark estimation**. *There are lots of ways to do this, but we are going to use the approach invented in 2014 by Vahid Kazemi and Josephine Sullivan.*

The basic idea is we will come up with 68 specific points (called landmarks) *that exist on every face — the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then we will train a machine learning algorithm to be able to find these 68 specific points on any face:*

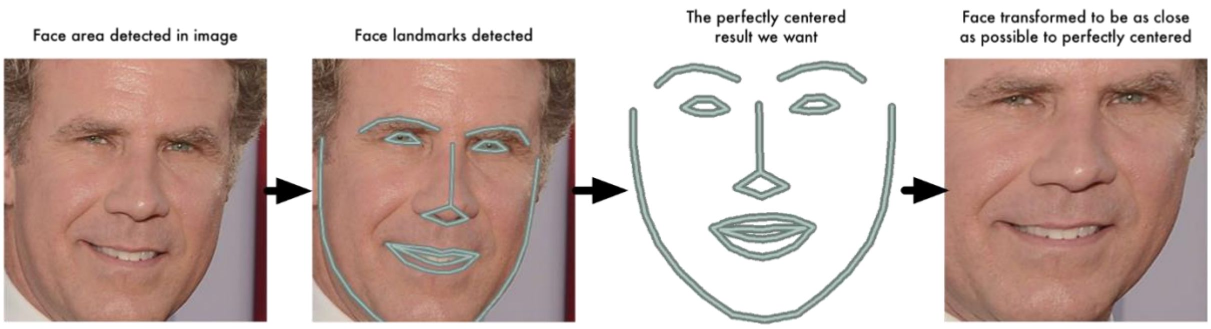


The 68 landmarks we will locate on every face. This image was created by [Brandon Amos]() of CMU who works on [OpenFace]().

Here's the result of locating the 68 face landmarks on our test image:



Now that we know were the eyes and mouth are, we'll simply rotate, scale and shear the image so that the eyes and mouth are centered as best as possible. We won't do any fancy 3d warps because that would introduce distortions into the image. We are only going to use basic image transformations like rotation and scale that preserve parallel lines (called affine transformations):


Face area detected in image
Face landmarks detected
The perfectly centered result we want
Face transformed to be as close as possible to perfectly centered

Now no matter how the face is turned, we are able to center the eyes and mouth are in roughly the same position in the image. This will make our next step a lot more accurate.

If you want to try this step out yourself using Python and dlib, here's the code for finding face landmarks and here's the code for transforming the image using those landmarks.

Step 3: Encoding faces

Now we are to the meat of the problem — actually telling faces apart. This is where things get really interesting!

The simplest approach to face recognition is to directly compare the unknown face we found in Step 2 with all the pictures we have of people that have already been tagged. When we find a previously tagged face that looks very similar to our unknown face, it must be the same person. Seems like a pretty good idea, right?

There's actually a huge problem with that approach. A site like Facebook with billions of users and a trillion photos can't possibly loop through every previous-tagged face to compare it to every newly uploaded picture. That would take way too long. They need to be able to recognize faces in milliseconds, not hours.

What we need is a way to extract a few basic measurements from each face. Then we could measure our unknown face the same way and find the known face with the closest measurements. For example, we might measure the size of each ear, the spacing between the eyes, the length of the nose, etc. If you've ever watched a bad crime show like CSI, you know what I am talking about

The most reliable way to measure a face

Ok, so which measurements should we collect from each face to build our known face database? Ear size? Nose length? Eye color? Something else?

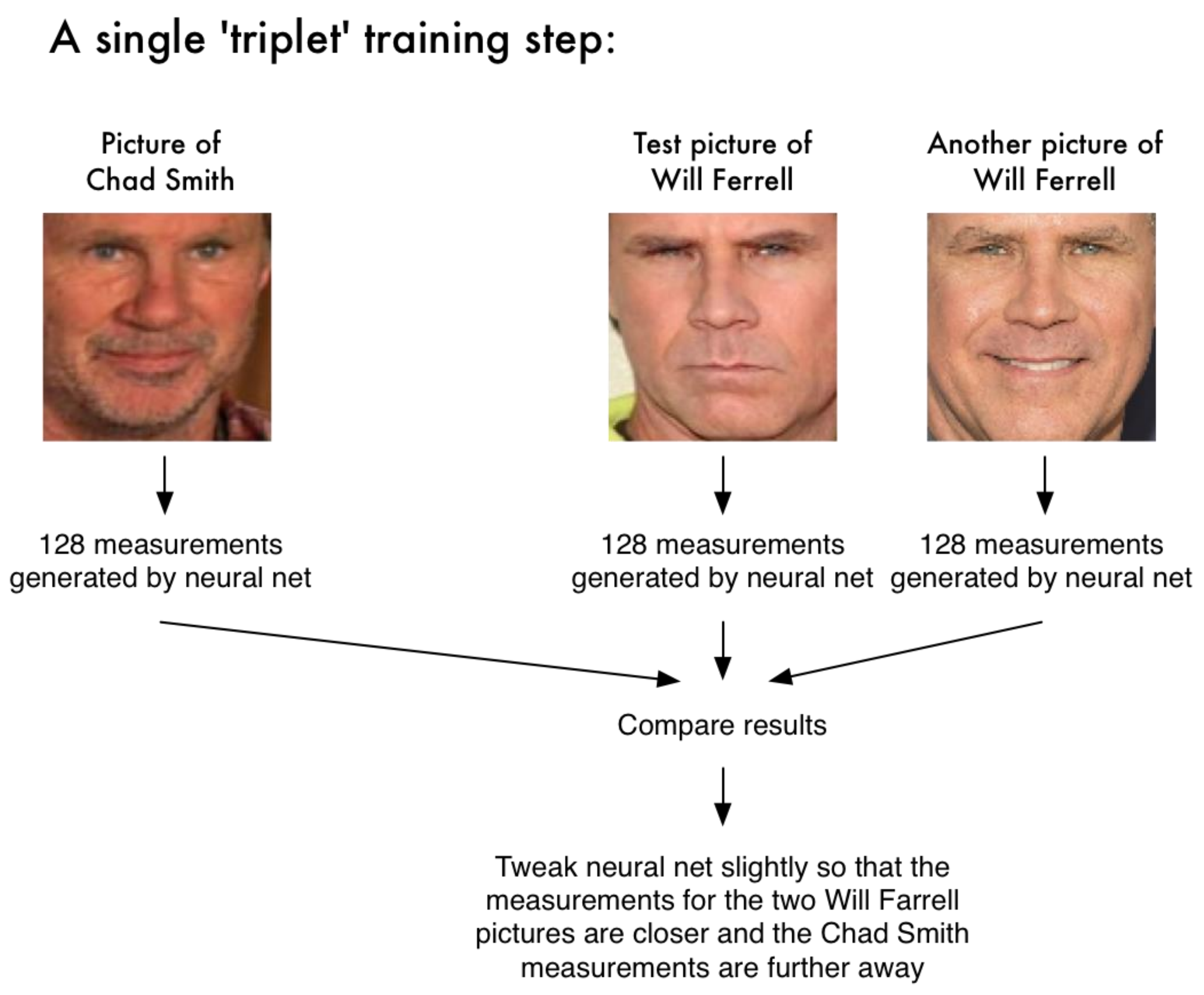
It turns out that the measurements that seem obvious to us humans (like eye color) don't really make sense to a computer looking at individual pixels in an image. Researchers have discovered that the most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

The solution is to train a Deep Convolutional Neural Network (just like we did in Part 3). But instead of training the network to recognize pictures objects like we did last time, we are going to train it to generate 128 measurements for each face.

The training process works by looking at 3 face images at a time:

1. *Load a training face image of a known person*
2. *Load another picture of the same known person*
3. *Load a picture of a totally different person*

Then the algorithm looks at the measurements it is currently generating for each of those three images. It then tweaks the neural network slightly so that it makes sure the measurements it generates for #1 and #2 are slightly closer while making sure the measurements for #2 and #3 are slightly further apart


A single 'triplet' training step:
Picture of Chad Smith
Test picture of Will Ferrell
Another picture of Will Ferrell
128 measurements generated by neural net
128 measurements generated by neural net
128 measurements generated by neural net
Compare results
Tweak neural net slightly so that the measurements for the two Will Farrell pictures are closer and the Chad Smith measurements are further away

After repeating this step millions of times for millions of images of thousands of different people, the neural network learns to reliably generate 128 measurements for each person. Any ten different pictures of the same person should give roughly the same measurements.

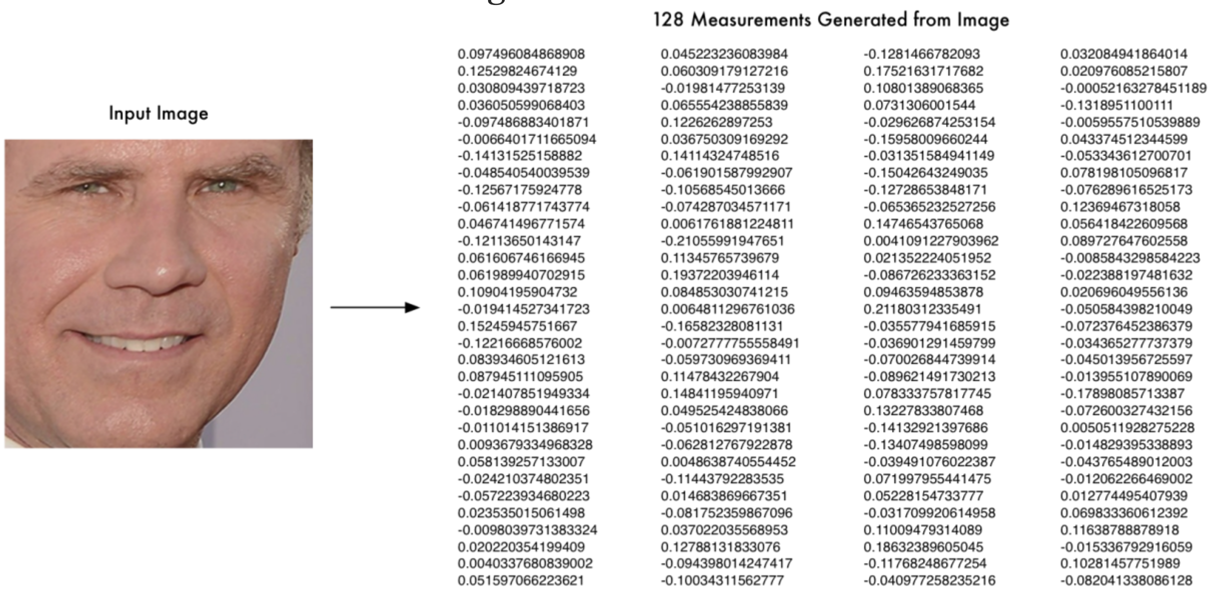
Machine learning people call the 128 measurements of each face an ***embedding****. The idea of reducing complicated raw data like a picture into a list of computer-generated numbers comes up a lot in machine learning (especially in language translation). The exact approach for faces we are using was invented in 2015 by researchers at Google but many similar approaches exist.*

Encoding our face image

This process of training a convolutional neural network to output face embeddings requires a lot of data and computer power. Even with an expensive NVidia Telsa video card, it takes about 24 hours of continuous training to get good accuracy.

But once the network has been trained, it can generate measurements for any face, even ones it has never seen before! So this step only needs to be done once. Lucky for us, the fine folks at OpenFace already did this and they published several trained networks which we can directly use. Thanks Brandon Amos and team!

So all we need to do ourselves is run our face images through their pre-trained network to get the 128 measurements for each face. Here's the measurements for our test image


Input Image

128 Measurements Generated from Image

0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

So what parts of the face are these 128 numbers measuring exactly? It turns out that we have no idea. It doesn't really matter to us. All that we care is that the network generates nearly the same numbers when looking at two different pictures of the same person.

If you want to try this step yourself, Open Face provides a lua script that will generate embeddings all images in a folder and write them to a csv file. You run it like this.

Step 4: Finding the person's name from the encoding

This last step is actually the easiest step in the whole process. All we have to do is find the person in our database of known people who has the closest measurements to our test image.

You can do that by using any basic machine learning classification algorithm. No fancy deep learning tricks are needed. We'll use a simple linear SVM classifier, but lots of classification algorithms could work.

All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person!

Let's review the steps we followed:

1. *Encode a picture using the HOG 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.*

3. *Pass the centered face image through a neural network that knows how to measure features of the face. Save those 128 measurements.*

4. *Looking at all the faces we've measured in the past, see which person has the closest measurements to our face's measurements. That's our match!*

Now that you know how this all works, here's instructions from start-to-finish of how run this entire face recognition pipeline on your own computer

Source code:

Attendance.py

```
import cv2
import numpy
import numpy as np
import face_recognition
import os
from datetime import datetime

path = 'imagesattendance'
images = []
classnames = []
mylist = os.listdir(path)
print(mylist)
for cls in mylist:
    curimg = cv2.imread(f'{path}/{cls}')
    images.append(curimg)
    classnames.append(os.path.splitext(cls)[0])
print(classnames)

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

def markattendance(name):
    with open('attendence.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()
            datestring = now.strftime('%H:%M:%S')
            f.writelines(f'\n{name},{datestring}')

encodelistknown = findencodings(images)
print('encoding complete')

cap = cv2.VideoCapture(0)

while True:
    success, img = cap.read()
    imgS = cv2.resize(img,(0,0),None,0.25,0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

    facescurrent = face_recognition.face_locations(imgS)
    encodescurrent = face_recognition.face_encodings(imgS,facescurrent)

    for encodeface,faceloc in zip(encodescurrent,facescurrent):
        matches =
face_recognition.compare_faces(encodelistknown,encodeface)
        facedis =
face_recognition.face_distance(encodelistknown,encodeface)
        #print(facedis)
        matchindex = np.argmin(facedis)

        if matches[matchindex]:
            name = classnames[matchindex].upper()
            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)
            markattendance(name)

    cv2.imshow('webcam',img)
    cv2.waitKey(1)
```

sample input:

Images in database:

Elon musk.jpg

Bill gates.jpg

Jackma.jpg

Sample test images:

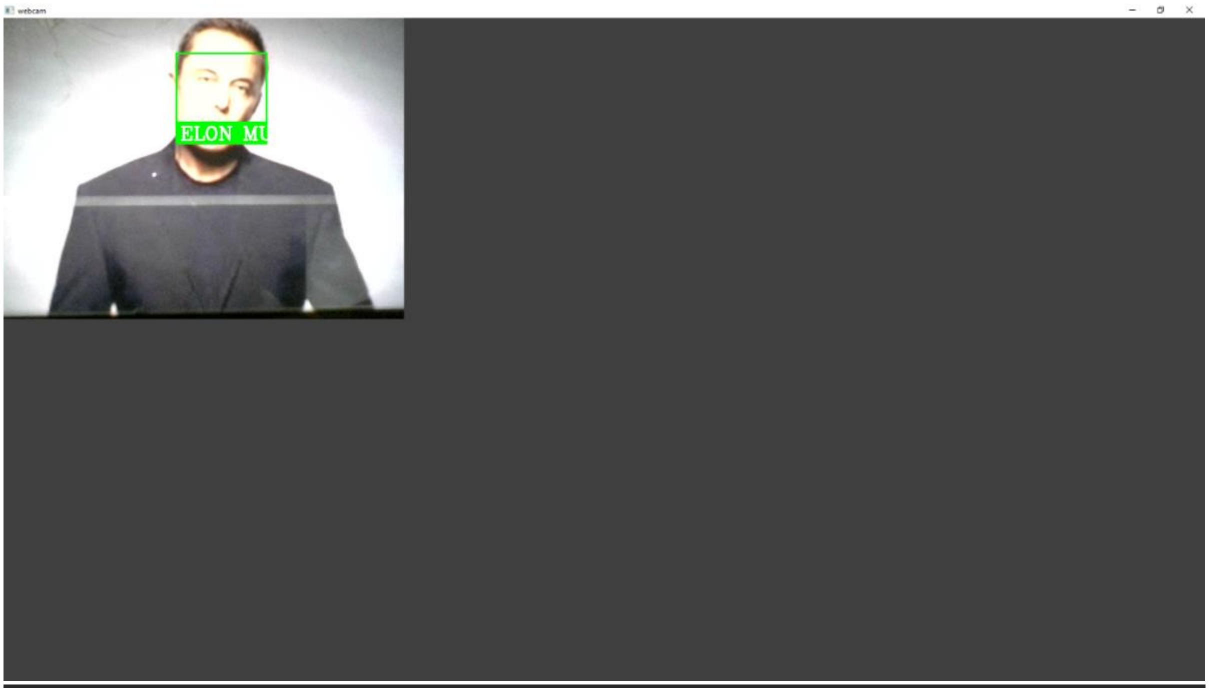
Elon musk test.jpg

Jackma test.jpg

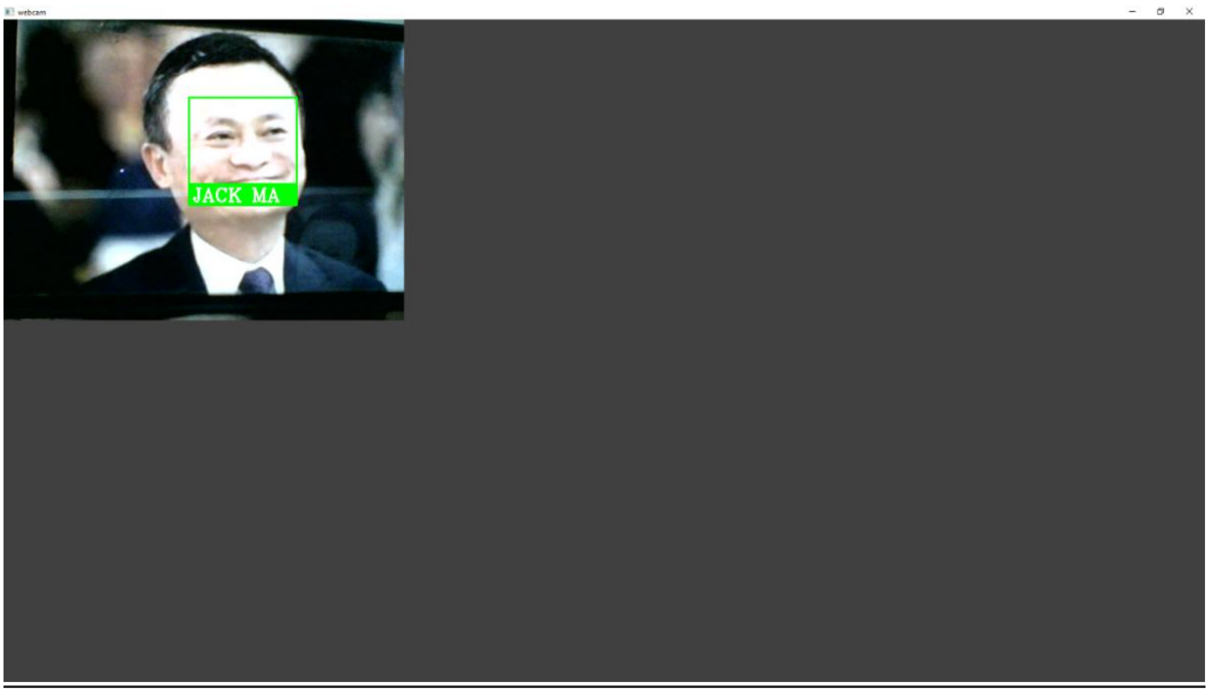
Bill gates test.jpg

Sample output:

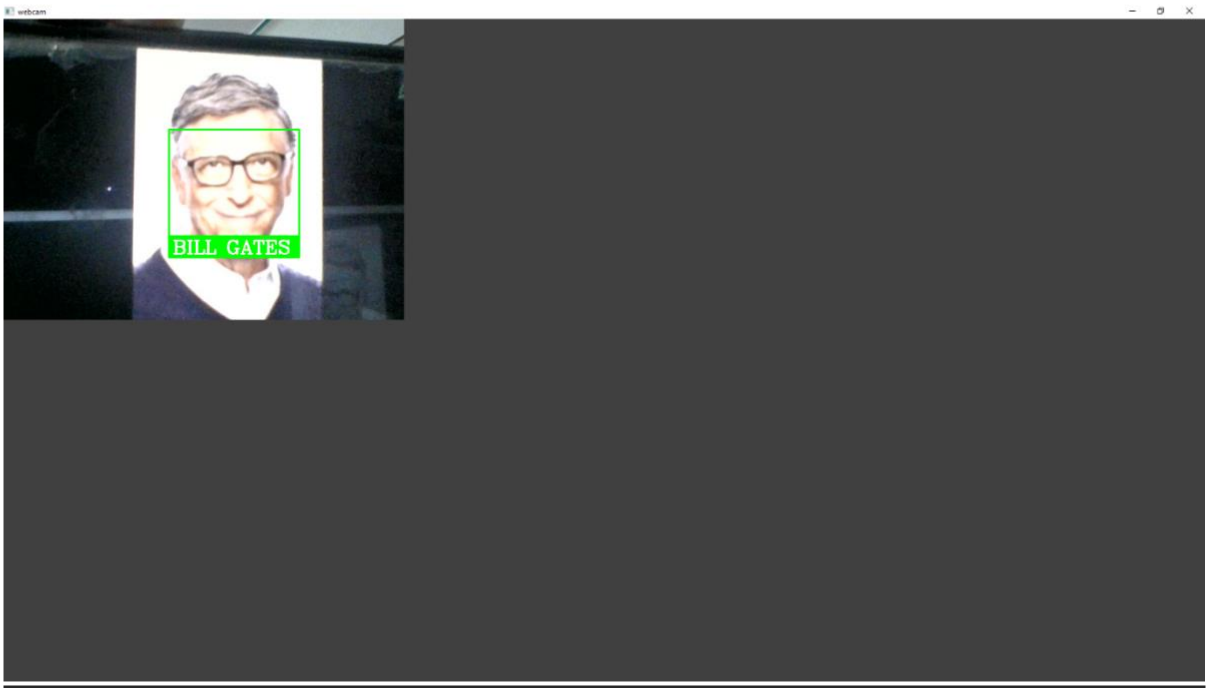
```
C:\Users\kishore\PycharmProjects\MINIPROJECT\venv\Scr
ipts\python.exe
"C:/Users/kishore/PycharmProjects/MINIPROJECT/attenda
nce project.py"
['kishore.jpg', 'elon musk.jpg', 'bill gates.jpg',
'jackma.jpg']
['kishore', 'elon musk', 'bill gates', 'jackma']
encoding complete
```



Elon musk output



Jackma output



Bill gates output

CONCLUSION:

This paper describes the mini-project for visual perception and autonomy module. Next, it explains the technologies used in the project and the methodology used. Finally, it shows the results, discuss the challenges and how they were resolved followed by a discussion. Using Haar-cascades for face detection worked extremely well even when subjects wore spectacles. Real time video speed was satisfactory as well devoid of noticeable frame lag. Considering all factors, LBPH combined with Haar-cascades can be implemented as a cost-effective face recognition platform. An example is a system to identify known troublemakers in a mall or a supermarket to provide the owner a warning to keep him alert or for automatic attendance taking in a class.