Real time class attendance tracker: applications of facial detection and recognition

Spriha Awasthi

12/5/2021

Introduction

In this project we try to build a real time class attendance tracker. This is built as an application of facial detection and recognition. We discuss some algorithms from literature that tackle these problems and use deep learning based embeddings to detect and identify faces. We will use open source tools to make implementation challenges and video processing easier. These tools have been discussed in the section 'Runtime Environment' towards the end.

Next we discuss the how the system is designed and managed.

Modeling attendance

The problem of marking a student as present or absent can be broken down as follows:

1. Capture a frame in video stream of class
2. Detect and recognize faces in the frame from known faces of students in class
3. If sufficient frames detect a person's face, mark their attendance as present.

This can be seen as a binomial problem with success being the student's attendance correctly recorded and failure being incorrectly recorded. Each trial in this distribution is a Bernoulli event of frame detection being correct which has success probaility 'p' as the accuracy of the model being used to detect and recognize face in the frame.

If we say that total 'N' frames are processed, then we can define attendance as :

```
if person A is detected in > d% of N frames:
        Mark them present
else:
        Mark them absent
```

Thus, if 1000 frames are processed and, d = 10, then if person A is detected in 101 frames they are marked present, else absent. By this approach of modeling entire process as a Binomial distribution the probability of a student's attendance being incorrectly marked becomes extremely low.

For example, if d = 10%, then absent is recorded if 90% or more frames fail to detect.

Say K = Ceil(0.9N), then Probability(Incorrect attendance) => Trial fails K or more times.

Thus, P(Incorrect attendance) = $\sum_{i=K}^{N} C_i^N * p^{N-i}(1-p)^i$

If d = 0, K = N, i.e present marked if even 1 frame detects attendance and then:

```
Probability(Incorrect attendance) = (1-p)^N
```

Say p = 50%, which is a very low classifier accuracy, and N = 100 frames are processed, then

```
Probability(Incorrect attendance) = 7.89 * 10^-31
```

So even lower accurate models can also give great results but false positives are bad. We overcome that problem by ensuring that d% of frames are detecting a student. The d% should be high enough to eliminate false positives and false negatives.

Let us now focus on each frame processing. Each frame will involve face detection, extraction and recognition.

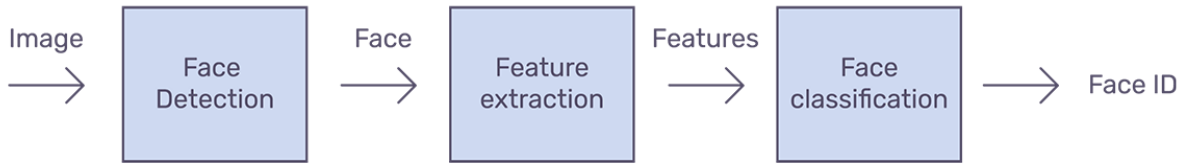


Figure 1: alt text

Datasets evaluation

There are different kinds of datasets at play in this project. The different classifiers each have different datasets as described below. A custom class specific dataset is created for real time attendance generation. Broadly there are 2 separate sections in which we are using different datasets: during different models of face detection and during facial features matching for recording attendance.

For face detection datasets and models, the Haar wavelets are not strictly based on any dataset but provided through OpenCV package. For the Histogram of Gradient + SVM approach we use trained models on the ibug 300-W dataset https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/.

Lastly, Histogram of Gradient + CNN model uses ResNet network with 29 convolution layers. The network was trained from scratch on a dataset of about 3 million faces. This dataset is derived from a number of datasets. The face scrub dataset http://vintage.winklerbros.net/facescrub.html, the VGG dataset http://www.robots.ox.ac.uk/~vgg/data/vgg_face/.

The real time face recognition is done by training on custom class specific dataset where the each student is represented by a folder with their name in the train/ folder. For example, train/Spriha/*.jpg would represent facial images for person named Spriha in the class. The training phase would generate the 128 dimension vector for each of these images per person and the 128 dimension cluster is formed for each person in the training dataset.

Each of the datasets is abiding by the FAIR principles. The datasets hosted at the links mentioned are findable at the hosted URLs. Except for the class specific data which is private to users and hosted/controlled by the administrator for personally identifiable data. They are also accessible as all but last are accessible at the hosted URLs without authentication. The class specific data is controlled by admin user's passwords.They are also interoperable and reusable as they are well documented and have fixed structure to be reused for different applications in different applications.

Face detection

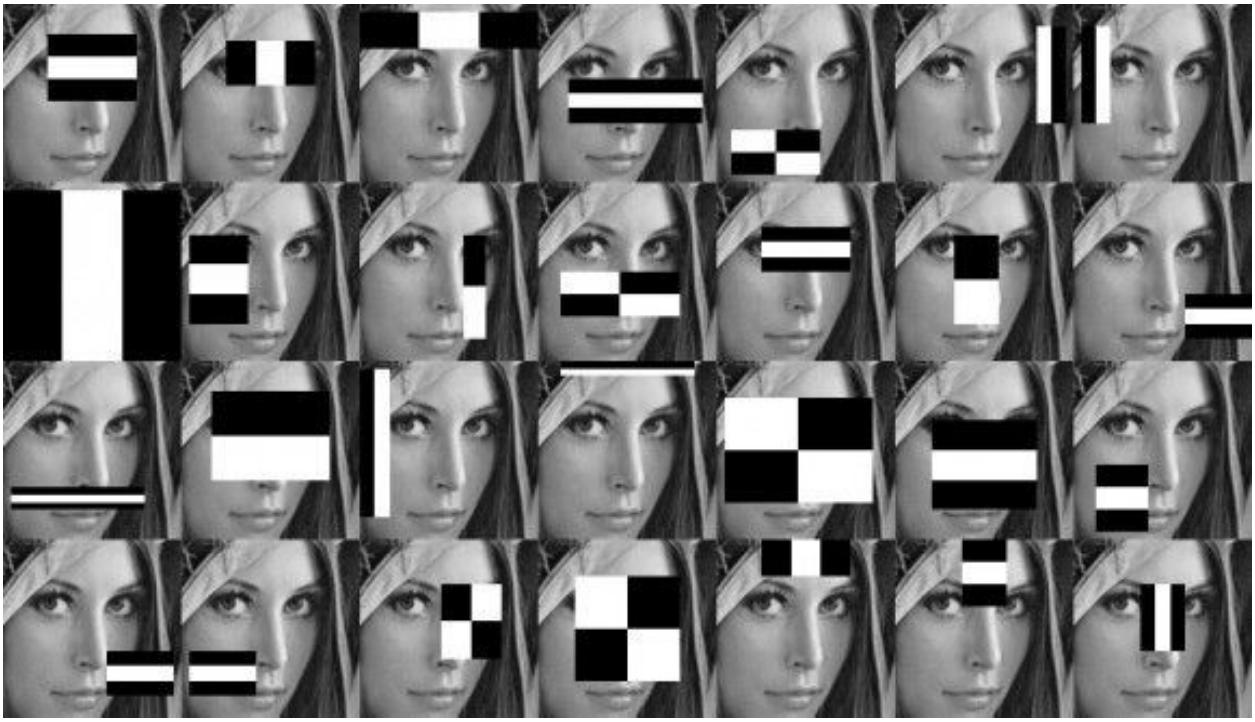
Face detection is a classical problem which has many established algorithms. We will consider a few classical ones and few more recent deep learning based approaches. For our use case, we want to pick an approach that is accurate but also fast enough to be executed in real time.

For our project, we have tested the following face recognition algorithms:

1. OpenCV based Haar Cascades
2. Histogram of Gradient + SVM
3. Histogram of Gradient + CNN

1) Open CV Haar Cascade

This method revolutionized face detection on devices. The method is blazing fast (e.g. on Macbook pro 15inc we have 38ms average frame per second processing) but less accurate (as low as 85% on local image set) as many false positives were detected while its implementation. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

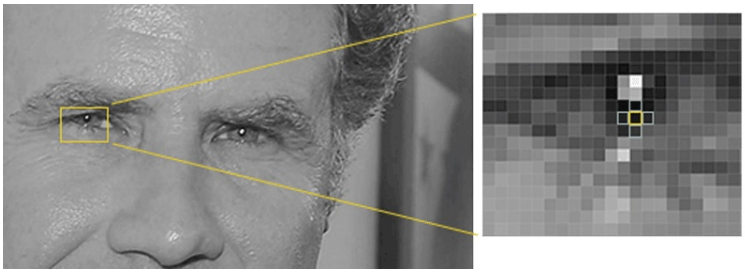
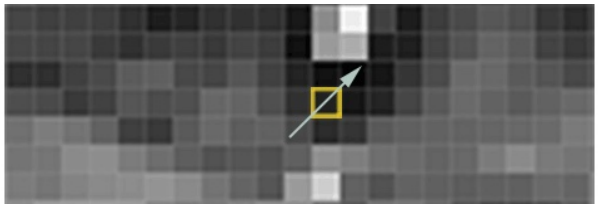


Haar Cascade being legacy algorithm require some advanced detection techniques with higher accuracy. To deal with lower accuracy and false positives of Haar, we will use HOG based SVM and CNN models provided by DLib open source tool.

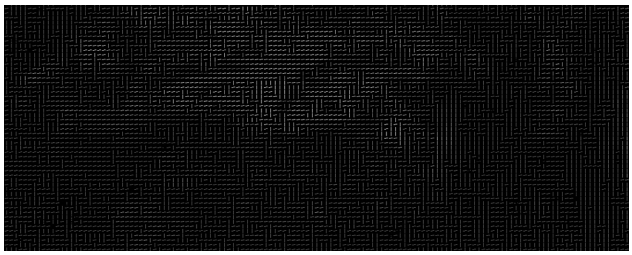
2) Histogram of gradients (HOG)

To find HOG of a frame in an image, we'll start by converting our colored image to gray scale because we don't need color data to find gradients for our use case. Lets start with the image below of Will Ferrel.

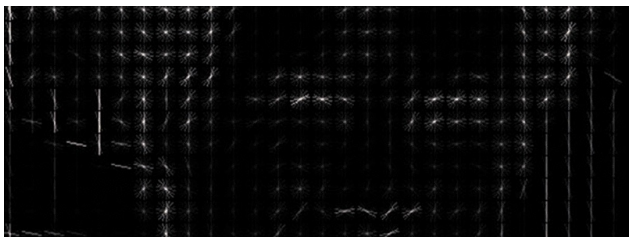
Then we'll look at every single pixel in our image one at a time. For every single pixel, we want to look at the pixels that directly surrounding it to compute gradient. Our goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then we want to draw an arrow showing in which direction the image is getting darker. Example in images below.

Eye subframe zoomed	Gradient at the center
	

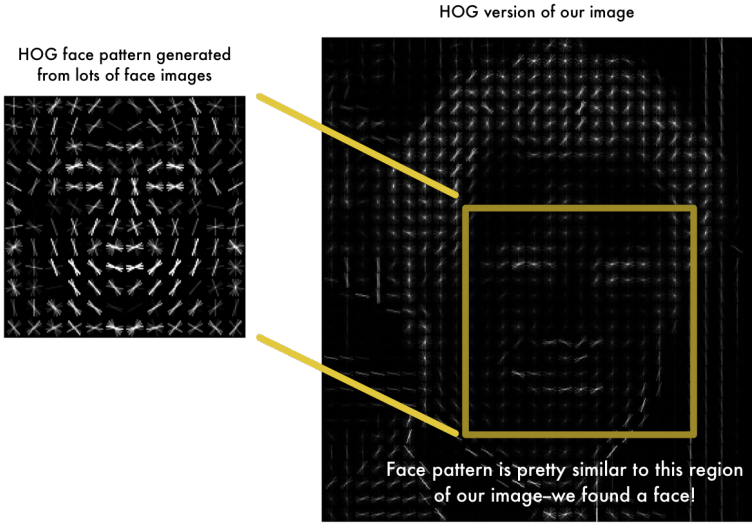
If we repeat this process for every single pixel in the image, we end up with every pixel being replaced by an arrow. These arrows are called gradients and they show the flow from light to dark across the entire image as below.

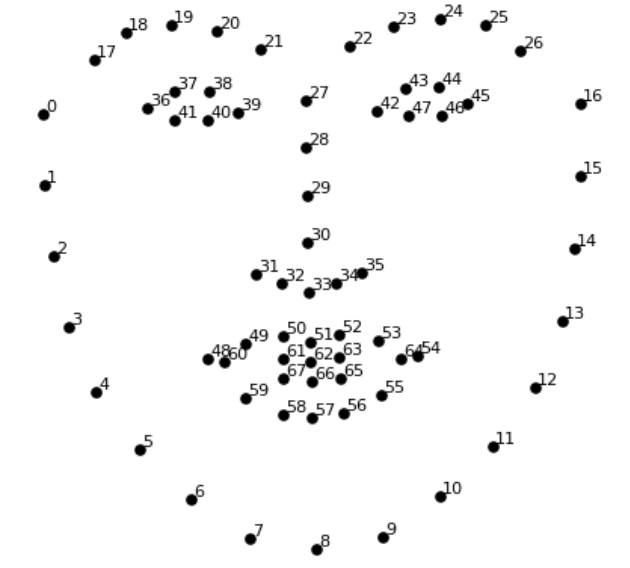
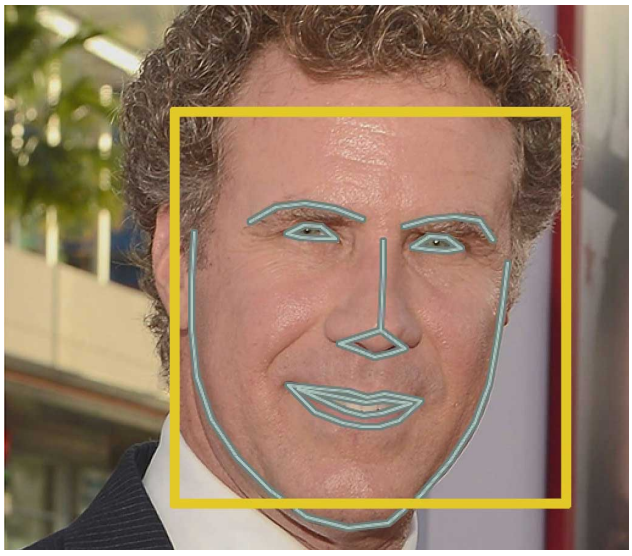
Original eye subframe	HOG Transformation matrix
	

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.

Complete eye zone	HOG Transformation
	

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:



68 Face landmarks	Observed results
	

To figure out which subframe of this HOG representation is a face, we can use 2 models which are pretrained and available in opensource library dlib. These are:

1. Default SVM approach which is trained on ibug dataset of 4000+ images to detect the face landmarks. 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.
2. Resnet34 based CNN which is trained on 3 million images. This is very powerful but consequently slow.

On Macbook pro machine the SVM based classifier takes 383ms per frame on average and CNN based approach takes 1890ms on average per frame. Let us write some code to compare results.

Below script will generate a face rectangle marked on the input image using HOG+SVM Approach.

```
from PIL import Image
import face_recognition
import cv2

imname = "friends_banner.jpeg"

# Load the jpg file into a numpy array
image = face_recognition.load_image_file(imname)

# Find all the faces in the image using the default HOG+SVM model.
face_locations = face_recognition.face_locations(image)

print("Found {} face(s) in this photograph.".format(len(face_locations)))
image = cv2.imread(imname)
for (i, face_location) in enumerate(face_locations):

    # Print the location of each face in this image
    top, right, bottom, left = face_location
    print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {}"\
        .format(top, left, bottom, right))

    # You can access the actual face itself like this:
    face_image = image[top:bottom, left:right]
    pil_image = Image.fromarray(face_image)
    pil_image.show()

    cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 2)
    # show the face number
    cv2.putText(image, "Face #{}".format(i + 1), (left - 10, top - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

cv2.imwrite('hog_'+imname, image)
```

Below script will generate a face rectangle marked on the input image using HOG+CNN Approach.

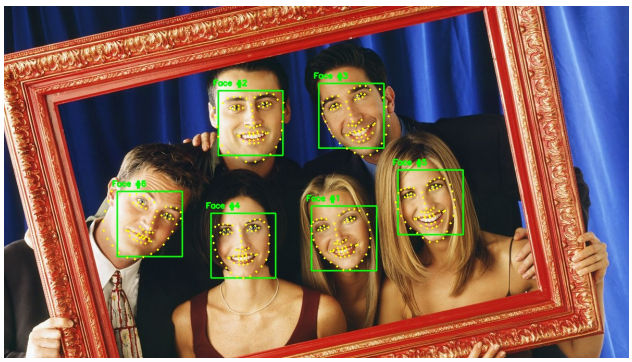
```
from PIL import Image
import face_recognition
import cv2
import matplotlib.pyplot as plt

imname = "friends_banner.jpeg"

# Load the jpg file into a numpy array
```

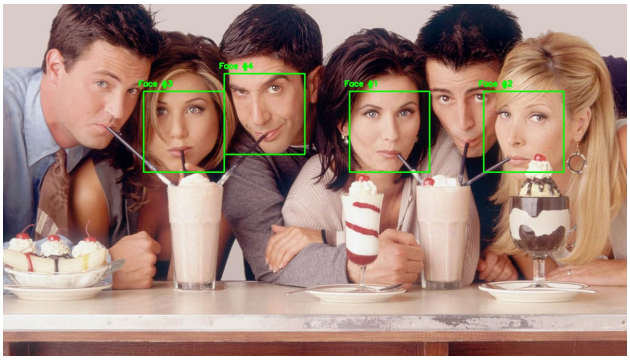
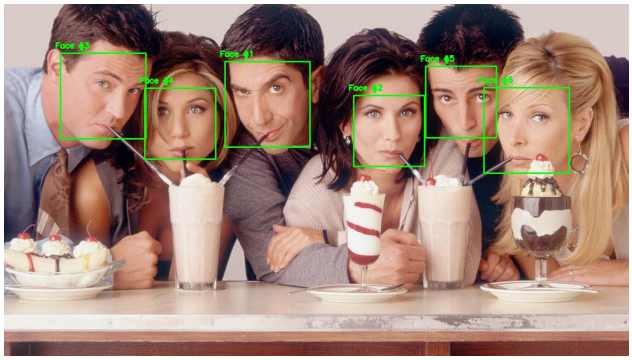
```
image = face_recognition.load_image_file(imname)

# Find all the faces in the image using convolutional neural network.
# This method is more accurate than the default HOG model, but it's slower
# unless you have an nvidia GPU and dlib compiled with CUDA extensions.
face_locations = face_recognition.face_locations(image, number_of_times_to_upsample=0,\
     model="cnn")

print("Found {} face(s) in this photograph.".format(len(face_locations)))
image = cv2.imread(imname)
for (i, face_location) in enumerate(face_locations):

    # Print the location of each face in this image
    top, right, bottom, left = face_location
    print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {}"\
        .format(top, left, bottom, right))

    # You can access the actual face itself like this:
    face_image = image[top:bottom, left:right]
    # pil_image = Image.fromarray(face_image)
    # pil_image.show()
    plt.imshow(face_image)

    cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 2)
    # show the face number
    cv2.putText(image, "Face #{}".format(i + 1), (left - 10, top - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)


cv2.imwrite('cnn_'+imname, image)
```

We can execute these as follows:

```
# The problem assumes a virtual runtime environment as described
# in section 'Runtime environment' later.

# source ./venv/bin/activate
# python find_faces_in_picture_cnn.py
```

The results of running the above code on 2 images is as below:

Original Image	Faces detected with HOG + SVM works
	

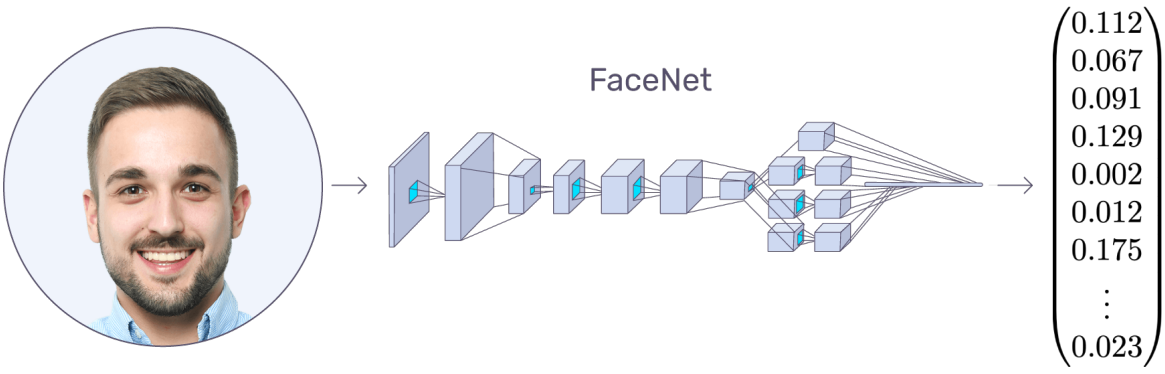
Faces with HOG+SVM less accurate	Faces with HOG+CNN wins
	

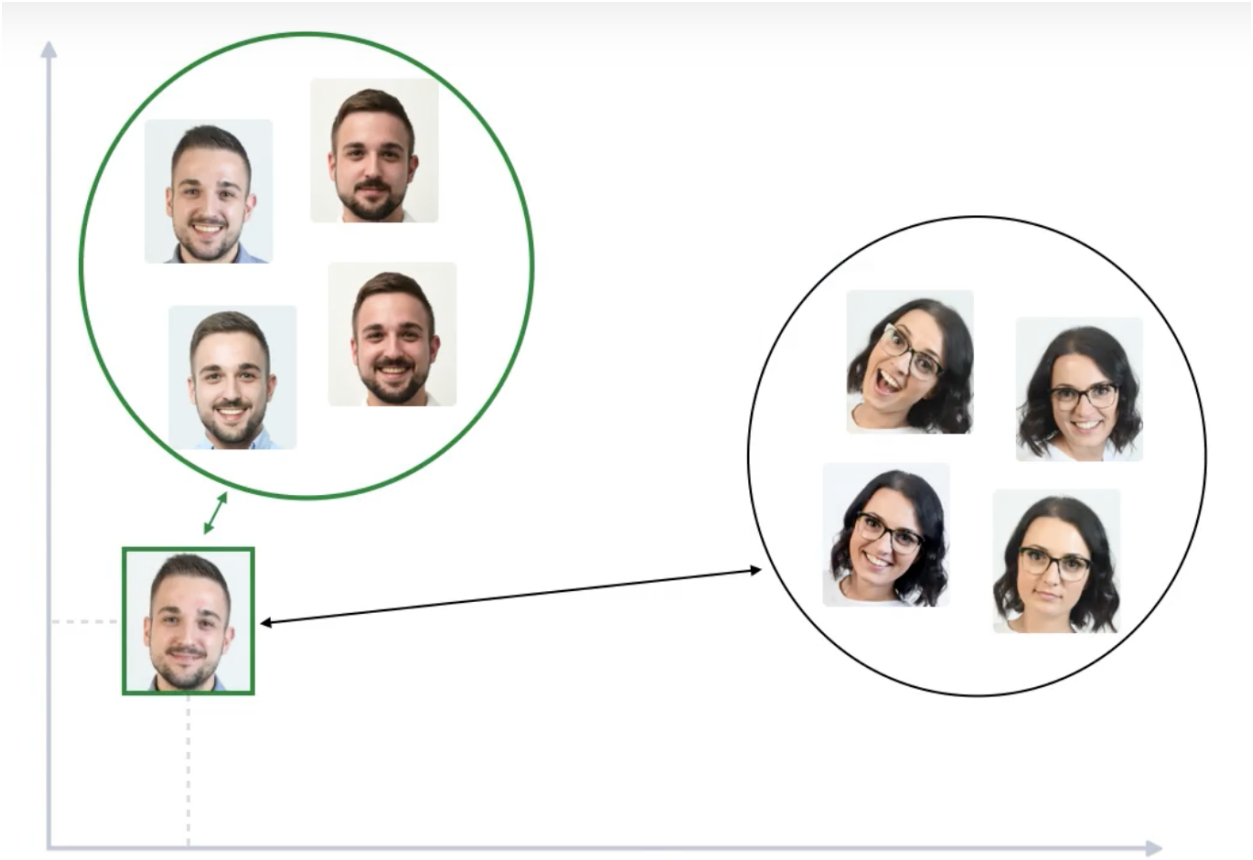
The above results show that SVM based approach is lower accurate when faces are turned but as proposed in the modeling section of attendance on multiple frames, due to lower time processing of SVM approach we will use that in our attendance tracker.

Face recognition

Now that face detection has been done, we can easily extract the subframe of the faces and for each frame generate a feature vector. For our use case we will use Facenet based algorithms 128 dimension vector for each face subframe.

Similarly, we collect for each student, a set of images for their faces and generate the 128 dimensional encodings for each student and create our training model. This approach is preferable as it is simple and works really well in realtime.





With the training data at hand we have a 128 dimensional space and we can use K-NN based approach or even simple euclidean distance comparisons with trained model encodings to decide which person is detected in the facial subframe from previous step.

Runtime environment

The heart of any applied modern machine learning solution is the infrastructure it is built on and the runtime environment. The tools and technologies used could play a vital role so that same solutions aren't rebuilt from scratch. One of the key learnings for me through this project was this understanding. For this project we use a variety of machine learning and assisting technologies to handle image/video data. Some of the libraries we used are below:

1. Install pyenv and install python 3.8.12 (https://github.com/pyenv/pyenv for your environment)
    a. brew install pyenv
    b. echo 'eval "$(pyenv init –path)"' » ~/.zprofile
    c. echo 'eval "$(pyenv init -)"' » ~/.zshrc
2. Make project directory and setup venv
    a. mkdir ~/Project
    b. cd Project
    c. pyenv local 3.8.12
    d. python -m venv venv
    e. source ./venv/bin/activate
    f. deactivate (when done)
3. Install tools to build and setup tools:
    a. brew install cmake
    b. pip install numpy
    c. pip install opencv-contrib-python
    d. pip install dlib
    e. pip install –upgrade imutils
    f. pip install face_recognition
    g. pip install matplotlib

The code and results are based on this runtime environment setup successfully and configured.

Demo scripts

Next we present some demo scripts for verifying the above findings.

Detect faces with Haar Cascade

```
import cv2
import sys
from time import time

# Average FPS : 0.03842847405410395 in frames: 205

cascPath = './haarcascade_frontalface_default.xml'
faceCascade = cv2.CascadeClassifier(cascPath)

video_capture = cv2.VideoCapture(0)
```

```
t = 0.0
n = 0
while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    start = time()

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

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE
    )

    t = t + time() - start
    n = n + 1

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

print("Average time per frame : " + str(t/n) + " in frames: " + str(n))

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Detect faces with HOG+SVM

```
import cv2
import sys
from time import time
import face_recognition

#Average FPS : 0.3839124798774719 in frames: 60 - HOG

video_capture = cv2.VideoCapture(0)

t = 0.0
n = 0
while True:
    # Capture frame-by-frame
```

```
    ret, image = video_capture.read()

    start = time()

    # Find all the faces in the image using the default HOG-based model.
    # This method is fairly accurate, but not as accurate as the CNN model and not GPU
    # accelerated.
    face_locations = face_recognition.face_locations(image)

    t = t + time() - start
    n = n + 1

    print("I found {} face(s) in this photograph.".format(len(face_locations)))
    for (i, face_location) in enumerate(face_locations):

        # Print the location of each face in this image
        top, right, bottom, left = face_location
        print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {}"\
            .format(top, left, bottom, right))

        cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 2)
        # show the face number
        cv2.putText(image, "Face #{}".format(i + 1), (left - 10, top - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', image)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

print("Average time per frame : " + str(t/n) + " in frames: " + str(n))

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Detect faces with HOG+CNN

```
import cv2
import sys
from time import time
import face_recognition

# Average FPS : 1.8839163621266684 in frames: 60 - CNN

video_capture = cv2.VideoCapture(0)

t = 0.0
n = 0
while True:
    # Capture frame-by-frame
```

```
    ret, image = video_capture.read()

    start = time()

    # Find all the faces in the image using the default HOG-based model that is provided by
    # dlib and face recognition library on top of it.
    face_locations = face_recognition.face_locations(image, number_of_times_to_upsample=0, \
        model="cnn")

    t = t + time() - start
    n = n + 1

    print("I found {} face(s) in this photograph.".format(len(face_locations)))
    for (i, face_location) in enumerate(face_locations):

        # Print the location of each face in this image
        top, right, bottom, left = face_location
        print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {}"\
            .format(top, left, bottom, right))

        cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 2)
        # show the face number
        cv2.putText(image, "Face #{}".format(i + 1), (left - 10, top - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', image)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

print("Average time per frame : " + str(t/n) + " in frames: " + str(n))

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Real time face recognition and attendance marker

Based on above principles here is the script for the real time facial detection. The training is clubbed in this script for simplicity of demo and understanding. However, on larger scale training on data of class images can be migrated to a separate script for batch processing and saving for later usage.

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

video_capture = cv2.VideoCapture(0)
DISTANCE_THRESHOLD = 0.55
PRESENCE_THRESHOLD = 0

COMPRESS_RATIO = 0.25
```

```
# Training on known faces. These are written here for simplicity of understanding
# but when class data is available, we can move this to a separate script.

# Load a sample picture and learn how to recognize it.
spriha_image = face_recognition.load_image_file("train/Spriha/Spa.jpeg")
spriha_face_encoding = face_recognition.face_encodings(spriha_image)[0]

# Load a second sample picture and learn how to recognize it.
sushma_image = face_recognition.load_image_file("train/Sushma/Sushma3.jpeg")
sushma_face_encoding = face_recognition.face_encodings(sushma_image)[0]

# Load a third sample picture and learn how to recognize it.
aaron_image = face_recognition.load_image_file("train/Aaron/Aaron.jpeg")
aaron_face_encoding = face_recognition.face_encodings(aaron_image)[0]

# Load a fourth sample picture and learn how to recognize it.
atharva_image = face_recognition.load_image_file("train/Atharva/Atharva.jpeg")
atharva_face_encoding = face_recognition.face_encodings(atharva_image)[0]


# Create arrays of known face encodings and their names. This can be saved to a file
# when training is done in separate script on large scale.

known_face_encodings = [
    spriha_face_encoding,
    sushma_face_encoding,
    aaron_face_encoding,
    atharva_face_encoding
]
known_face_names = [
    "Spriha",
    "Sushma",
    "Aaron",
    "Atharva"
]

# Initialize some variables
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

attendees = {}
current_count = 0
n = 0

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Resize frame of video to compress size for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=COMPRESS_RATIO, fy=COMPRESS_RATIO)
```

```
    # Convert the image from BGR color (which OpenCV uses) to RGB color
    # (which face_recognition uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time
    if process_this_frame:
        n = n + 1

        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for name in attendees.keys():
            attendees[name]['present'] = 0

        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Unknown"

            # # If a match was found in known_face_encodings, just use the first one.
            # if True in matches:
            #     first_match_index = matches.index(True)
            #     name = known_face_names[first_match_index]

            # Or instead, use the known face with the smallest distance to the new face
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index] and face_distances[best_match_index] < DISTANCE_THRESHOLD:
                name = known_face_names[best_match_index]
                if name not in attendees:
                    attendees[name] = {'present': 0, 'count' : 0}
                attendees[name]['present'] = 1
                attendees[name]['count'] = attendees[name]['count'] + 1



            face_names.append(name)

    process_this_frame = not process_this_frame

    # Display the results
    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Scale back up face locations since the frame we detected in was scaled to 1/4 size
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        # Draw a box around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
```

```
        if name in attendees:
            attendees[name]['image'] = frame[top:bottom, left:right]

        # Draw a label with a name below the face
        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

    # Display the resulting image
    cv2.imshow('Video', frame)

    if current_count < len(attendees):
        current_count = len(attendees)
        print("Current attendance: " + str(current_count))

    # Hit 'q' on the keyboard to quit!
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release handle to the webcam
video_capture.release()
cv2.destroyWindow('Video')
print(attendees)
print('Total frames processed: ' + str(n))

# Generate a collage of current students attendees for run.
if attendees:
    images = []
    row = []
    count = 0
    for name in attendees:
        if attendees[name]['count'] / n > PRESENCE_THRESHOLD:
            frame = cv2.resize(attendees[name]['image'], (150, 150))

            # Draw a label with a name below the face
            cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

            if len(row) > 0:
                row = np.hstack([row, frame])
            else:
                row = frame

        count = count + 1
        if count % 10 == 0 or count == len(attendees):
            if len(images) > 0:
                images = np.vstack([images, row])
            else:
                images = row

            row = []
```

```
    cv2.imshow('Class attendees', images)
    cv2.waitKey(0)
    now = datetime.now()
    cv2.imwrite("attendance_" + now.strftime("%Y-%m-%d_%H_%M_%S") + ".jpeg", images)

cv2.destroyAllWindows()
```

The above code was demonstrated in class and is consolidated here for simplicity of reading.

Other resources & references

The project is hosted at https://github.com/sprihap/Projects/tree/master/realtime_attendance_tracker and also contains other scripts used for generating and comparing results we have discussed. We have used extensively several opensource project resources and want to acknowlege their immense contribution to learning:

1. OpenCV (https://opencv.org/)
2. Dlib (http://dlib.net/)
3. Face recognition, Numpy, SciKit python packages
4. https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/