Technical Report

Automatic Attendance Tracker through CV and deep learning

Submitted by:

Akhil - D19003
PG Diploma Data Science
Praxis Business School

Kailash - D19013
PG Diploma Data Science
Praxis Business School

Nikhil - D19019
PG Diploma Data Science
Praxis Business School

Nishant - D19020
PG Diploma Data Science
Praxis Business School

Prabhath-D19022
PG Diploma Data Science
Praxis Business School

Under the Guidance of
Prof. Gourab Nath
Praxis Business School

Index

Project Overview:

Maintaining attendance is very important in all the institutes for checking the performance of students, every institute has its own method in this regard. Attendance is of prime importance for both the teacher and student of an educational organization.
The problem arises when we think about the traditional process of taking attendance in the classroom. Calling name or roll the number of students for attendance not only wastes time but also requires energy.

So the installation of an automatic attendance system will solve all these problems. There are some automatic attendance taking systems which are currently being used by multiple institutions. An example of one such system is the use of biometric techniques. Although it is automatic and a step ahead of the traditional method, it fails to meet the time constraint. The student has to wait in the queue for giving attendance, which is time taking.

This project introduces facial detection and recognition methods to automatically generate attendance. Facial Technologies have undergone large scale upgrades in performance in the last decade and such systems are now popular in fields such as security and commerce. This work details an automated attendance system that will mark the attendance of students. The proposed system is a real-world solution to handle the day-day activities of an organization such as a college.

Problem Statement

This technical report describes a machine learning approach to automate facial recognition system which is capable of detecting images rapidly and achieving high detection rates. This report is distinguished by a series of experiments that directs two key contributions. The first is the introduction of a face detection algorithm which allows the features used by our model to compute results very quickly. The second is an image comparison algorithm, based on AWS Rekognition, which recognizes a small number of critical visual features from a larger set of images and yields extremely efficient predictions.

The goal is to automate the process of attendance in a classroom, with just a click of a classroom picture. The task involved are the following:

1. Download and pre-process the image.
2. Forwarding the processed image to the face detection algorithm.
3. Extracting all the faces and collecting their respective IDs.
4. Extracting unique features, classifying them, and storing.

5. Training the classifier that recognizes the face using Amazon Web Services.
6. Summarizing the results in an excel.

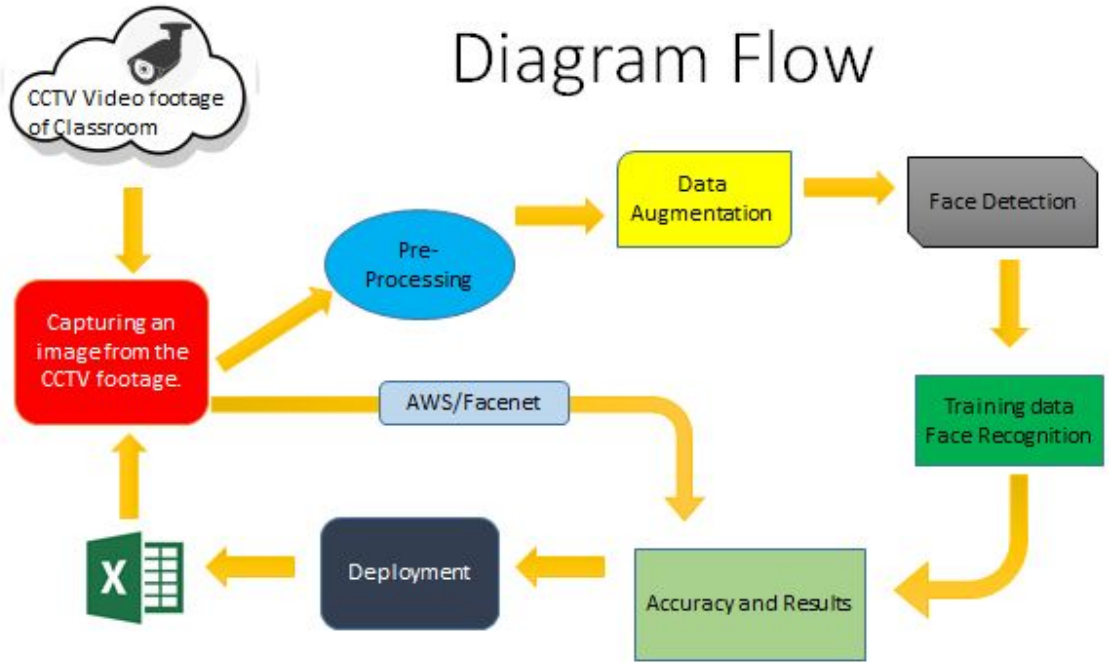
With detection and recognization in place, the excel contains the name of students and their attendance respectively.

Algorithms and Techniques

Face detection involves separating image windows into two classes; one containing faces and another containing the background (clutter). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin color and facial expression. The problem is further complicated by differing lighting conditions, image qualities, and geometries, as well as the possibility of partial occlusion and disguise. An ideal face detector would, therefore, be able to detect the presence of any face under any set of lighting conditions, upon any background.

The face detection task can be broken down into two steps. The first step is a classification task that takes some arbitrary image as input and outputs a binary value of yes or no, indicating whether there are any faces present in the image. The second step is the face localization task that aims to take an image as input and output the location of any face or faces within that image as some bounding box with (x, y, width, height).

Methodology



Below are the methodology and descriptions of the applications used for face detection, training, and face recognition.

Face Detection

Experiment 1:

AIM: Detect face using the Viola-Jones algorithm.

Method: Viola Jones

The first stage was creating a face detection system using Haar-cascades. Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001.

This approach for object detection was to minimize computation time while achieving high detection accuracy. Paul Viola and Michael Jones proposed a fast and robust method for face detection which is 15 times quicker than any technique at the time of release with 95% accuracy at around 17 fps. The technique relies on the use of simple Haar-like features that are evaluated

quickly through the use of new image representation. Based on the concept of an —Integral Image‖ it generates a large set of features and uses the boosting algorithm AdaBoost to reduce the overcomplete set and the introduction of a degenerative tree of the boosted classifiers provides for robust and fast interferences. The detector is applied in a scanning fashion and used on gray-scale images, the scanned window that is applied can also be scaled, as well as the features evaluated.

Implementation:

1. Initiated with new image representation called an *integral image* that allows for very fast feature evaluation.

2. Applied a method for constructing a classifier by selecting a small number of important features using *AdaBoost*.

3. The third major footstep was a method for combining successively more complex classifiers in a cascade structure which dramatically increases the speed of the detector by focusing attention on promising regions of the image.

Code:

```
#### Experimenting with scale factor
import cv2
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 300
classroom=cv2.imread(r'D:\Downloads\class.jpg')
gclass=cv2.cvtColor(classroom,cv2.COLOR_BGR2GRAY)
fc=cv2.CascadeClassifier('D:\\Software\\Anaconda\\Lib\\site-packages\\cv2\
\data\\haarcascade_frontalface_alt.xml')
scale=np.arange(1.001,1.201,0.002)
for i in scale:
    gclass=cv2.cvtColor(classroom,cv2.COLOR_BGR2GRAY)
    faces=fc.detectMultiScale(gclass,scaleFactor=i)
    print('Number of faces detected:',len(faces))
    for (x, y, w, h) in faces:
        # Draw rectangle around the face
        cv2.rectangle(gclass, (x, y), (x+w, y+h), (255, 255, 255), 3)
    plt.figure(figsize=(8,8))
    plt.imshow(gclass, cmap='gray')
    plt.show()
```

Hyper parameter tuning :

```
classroom=cv2.imread(r'D:\Downloads\class.jpg')
gclass=cv2.cvtColor(classroom,cv2.COLOR_BGR2GRAY)
fc=cv2.CascadeClassifier('D:\\Software\\Anaconda\\Lib\\site-packages\\cv2\
\data\\haarcascade_frontalface_alt.xml')
neigh=np.arange(1,11,1)
for i in neigh:
    gclass=cv2.cvtColor(classroom,cv2.COLOR_BGR2GRAY)
    faces=fc.detectMultiScale(gclass,minNeighbors=i)
    print('Number of faces detected:',len(faces))
    for (x, y, w, h) in faces:
        # Draw rectangle around the face
        cv2.rectangle(gclass, (x, y), (x+w, y+h), (255, 255, 255), 3)
    plt.figure(figsize=(8,8))
    plt.imshow(gclass, cmap='gray')
    plt.show()
```

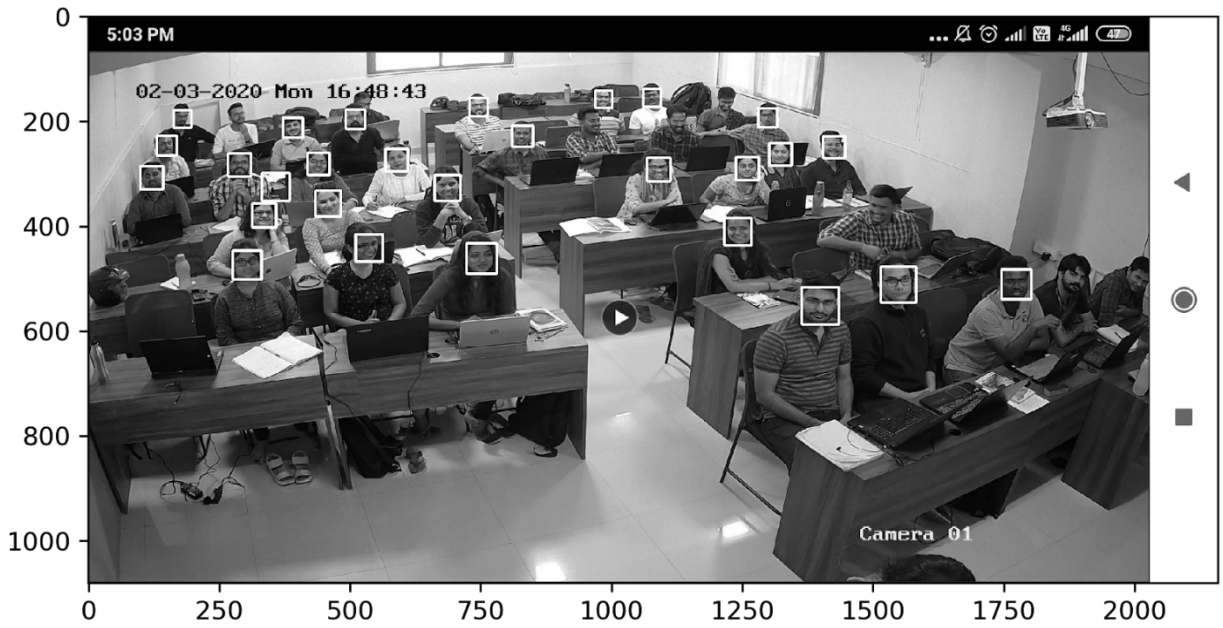
Grid Search:

```
classroom=cv2.imread(r'D:\Downloads\class.jpg')
gclass=cv2.cvtColor(classroom,cv2.COLOR_BGR2GRAY)
fc=cv2.CascadeClassifier('D:\\Software\\Anaconda\\Lib\\site-packages\\cv2\
\data\\haarcascade_frontalface_alt.xml')
neigh=np.arange(1,11,1)
scale=np.arange(1.001,1.152,0.002)
for i in scale:
    for j in neigh:
        gclass=cv2.cvtColor(classroom,cv2.COLOR_BGR2GRAY)

faces=fc.detectMultiScale(gclass,minNeighbors=j,scaleFactor=i,minSize=(25,
25))
        if ((len(faces)>=27)&(len(faces)<=32)):
            print(f'Scale:{i} ,MinNeighbour:{j}')
            print('Number of faces detected:',len(faces))
            for (x, y, w, h) in faces:
        # Draw rectangle around the face
```

```
                cv2.rectangle(gclass, (x, y), (x+w, y+h), (255, 255, 255),
3)
            plt.figure(figsize=(8,8))
            plt.imshow(gclass, cmap='gray')
            plt.show()
```

Results and Outcomes:



Research Paper:
Rapid Object Detection using a Boosted Cascade of Simple Features

References:

1. https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
2. https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999

Improvement in Experiment 1

1. Method: **Histogram Equalization**

Histogram Equalization is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast

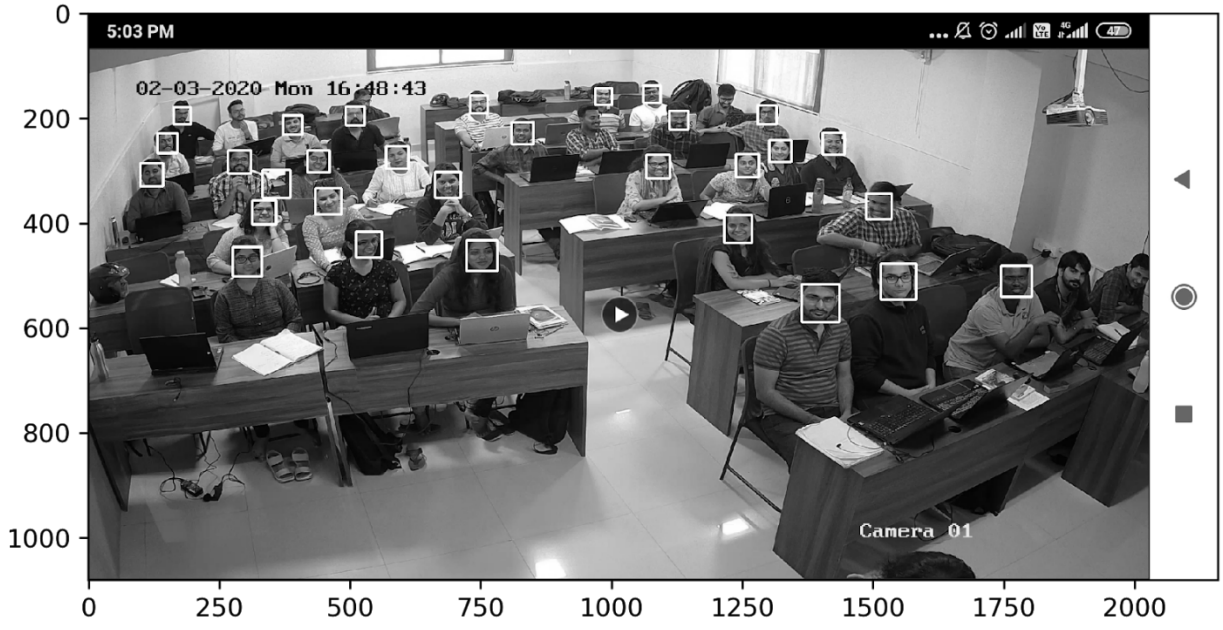
of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.

Histogram equalization is a more sophisticated technique, modifying the dynamic range of an image by altering the pixel values, guided by the intensity histogram of that image. Histogram equalization creates a non-linear mapping, which reassigns the intensity values in the input image such that the resultant images contain a uniform distribution of intensities, resulting in a flat (or nearly flat) histogram. This mapping operation is performed using a lookup table. The resulting image typically brings more image details to light, since it makes better use of the available dynamic range. The steps in the histogram equalization process are:

Implementation:

1. For simplicity, we are considering gray-scale images.
2. For Equalization: We first computed the histogram and then normalized it into a probability distribution.
3. For Normalization: We divided the frequency of each pixel intensity value by the total number of pixels present in the image (which was equal to the resolution of the image i.e(Nrows x Ncols).
4. Transfer the normalized histogram to a color table.
5. Applied transformation function for the image in order to obtain a flat histogram.
6. Transfer the input image through the lookup table.

Results and Outcomes:



Research Papers:
Face recognition based on improved BP neural network
References:

1. https://www.researchgate.net/publication/328908424_Face_Recognition_Based_on_Histogram_Equalization_and_Convolution_Neural_Network
2. https://towardsdatascience.com/histogram-equalization-5d1013626e64

2. Method: Grid search for hyperparameter optimization.

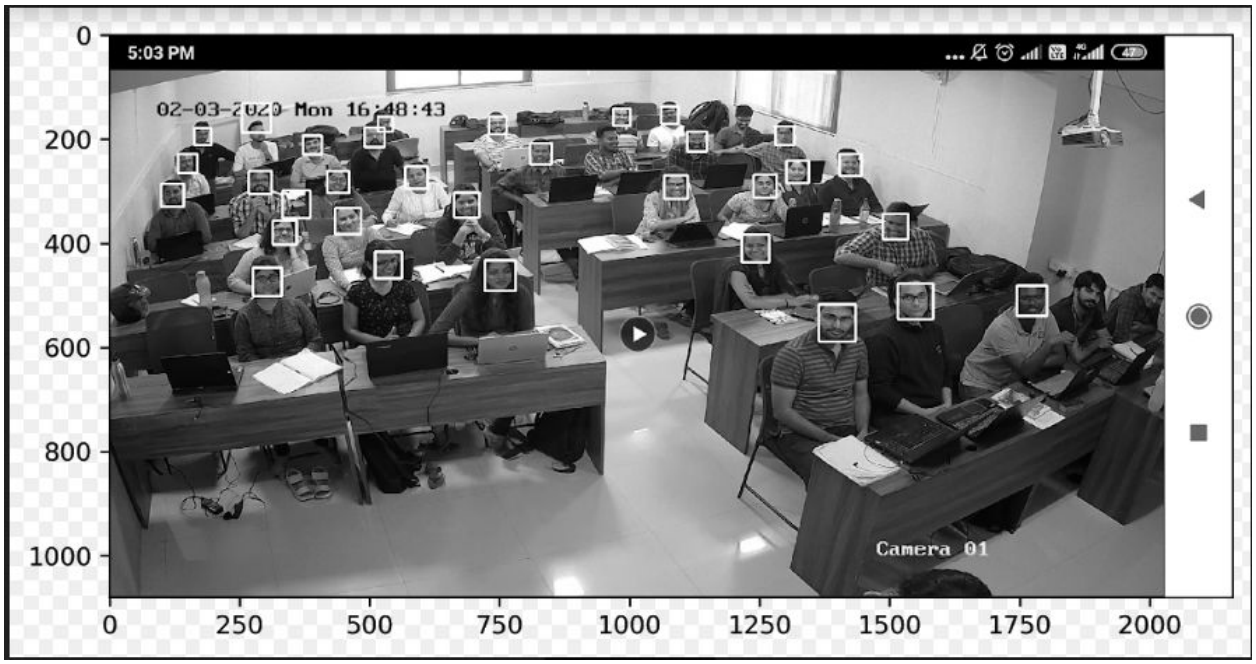
Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters. Using sklearn's *GridSearchCV*, we first define our grid of parameters to search over and then run the grid search.

Grid-search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions. This is significant as the performance of the entire model is based on the hyperparameter values specified.

Implementation:

1. We initiated by importing *GridSearchCV* from the sklearn library, which is a machine learning library for python.
2. The estimator parameter of GridSearchCV is used for the hyperparameter tuning process.
3. We have used the RBF kernel of the Support Vector Regression Model (SVR), and the parameters used were c, gamma, and epsilon.
4. We then provided a list of values to each hyperparameter of the model to choose from.
5. Changes these values and experimented more to observe which value ranges gave better performance.
6. Finally, a cross-validation process is performed to determine the hyperparameter value set for the best accuracy levels.

Results and Outcomes:



References:

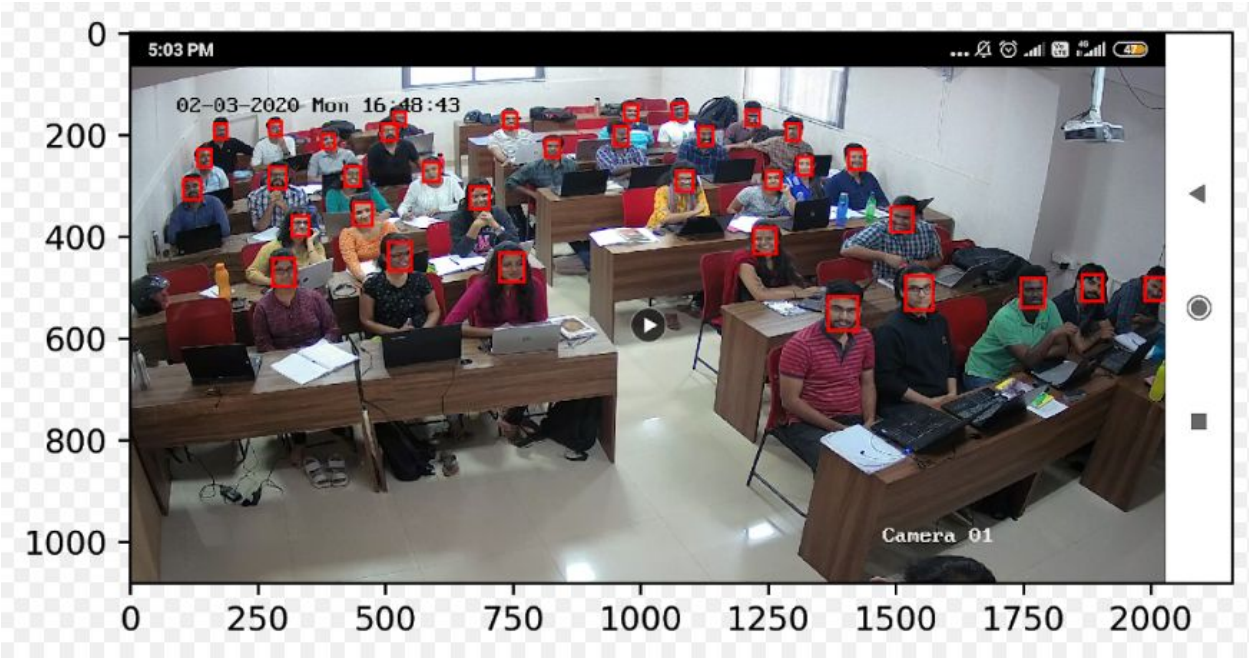
1. https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

Experiment 2:

Method: MTCNN (Multi-task Cascaded Convolutional Neural Networks)
MTCNN is an algorithm consisting of 3 stages, which detects the bounding boxes of faces in an image along with their 5 Point Face Landmarks. Each stage gradually improves the detection results bypassing its inputs through a CNN, which returns candidate bounding boxes with their scores, followed by non-max suppression.

In stage 1 the input image is scaled down multiple times to build an **image pyramid** and each scaled version of the image is passed through it's CNN. In stage 2 and 3 we extract image patches for each bounding box and resize them (**24x24** in stage 2 and **48x48** in stage 3) and forward them through the CNN of that stage. Besides bounding boxes and scores, stage 3 additionally computes **5 face landmarks points** for each bounding box.

Results and Outcomes:



Research Paper :

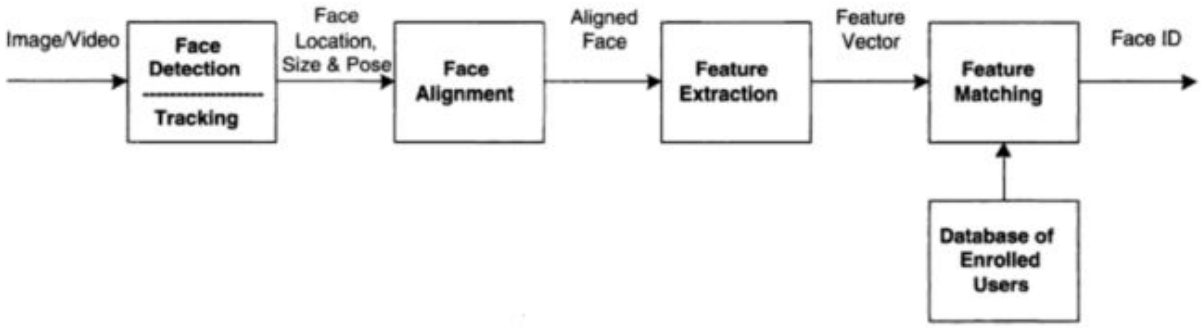
Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks

References:

1. https://arxiv.org/abs/1604.02878
2. https://towardsdatascience.com/how-does-a-face-detection-program-work-using-neural-networks-17896df8e6ff

Face Recognization

Here, after detection, we extract all the faces and if a particular person had his face captured and trained before, our recognizer will make a “prediction” returning its id and an index, showing how confident the recognizer is with this match.



Experiment 1:

AIM: Training on Images.

We have used RESNET Architecture to train our model.

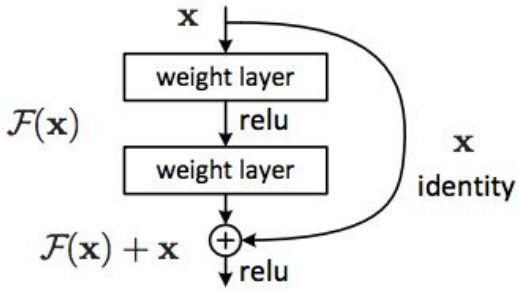
Why RESNET?

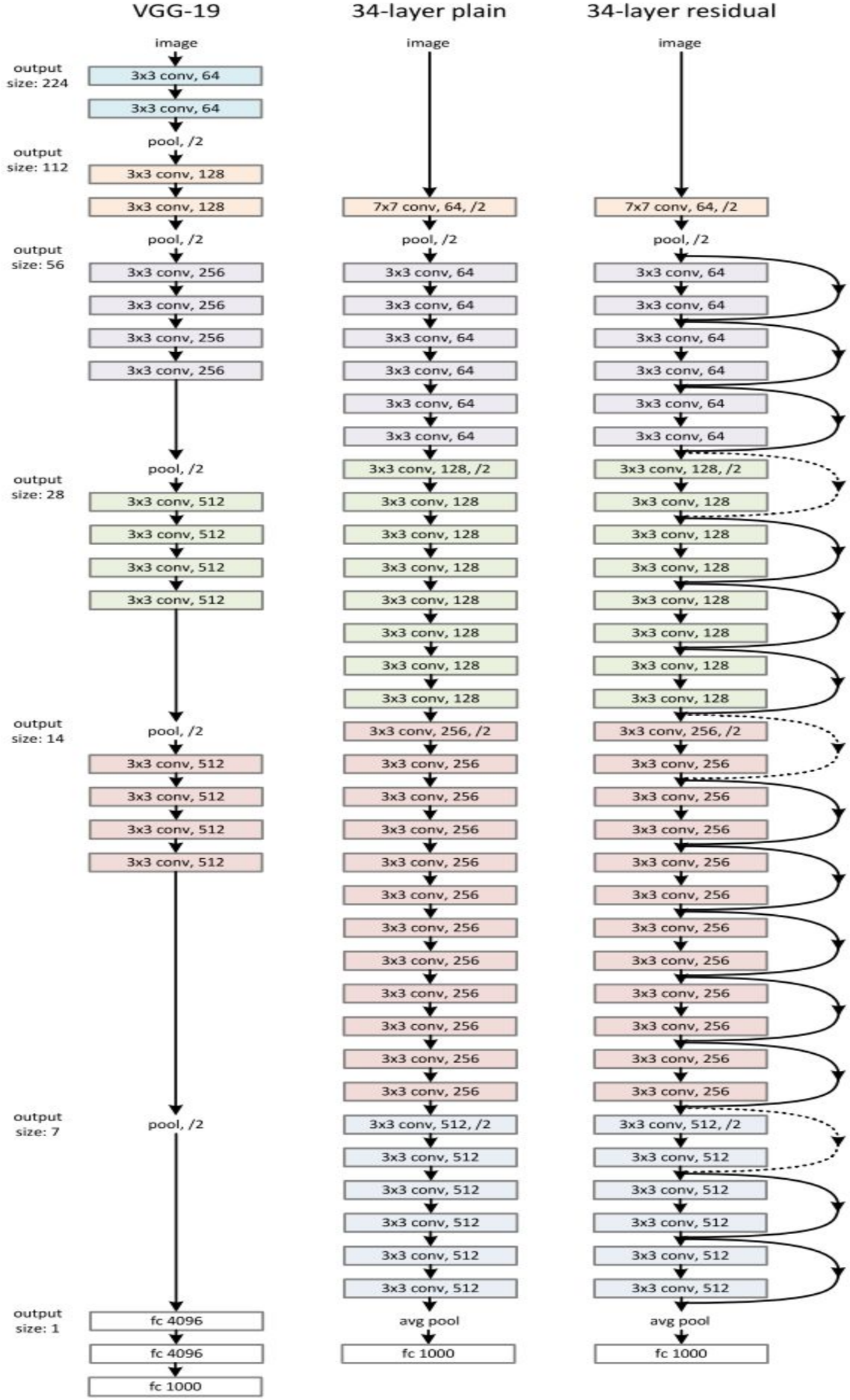
Deep convolutional networks have led to remarkable breakthroughs for image classification. Driven by the significance of convolutional neural networks, the residual network (ResNet) was created. In Resnet a model is built by his team which bagged the image classification challenges in all the domains such as classification, detection, and localization.

What Does ResNet Do?

The main goal of the residual network is to build a deeper neural network. We have two intuitions based on this:

1. As we keep going deeper into implementing a large number of layers, we make sure not to degrade the accuracy and error rate. This is handled by identity mapping.
2. Keep learning the residuals to match the predicted with the actual.




VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

References:
https://arxiv.org/abs/1512.03385

Improvement 1: Training with normal Images.

Improvement 2: Training with different Augmentations.

Method: Data Augmentations

Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Data Augmentation is applied so that they can learn features that are invariant to their location in the image. Nevertheless, augmentation can further aid in this transform invariant approach to learning and can aid the model in learning features that are also invariant to transforms such as left-to-right to top-to-bottom ordering, light levels in photographs, and more.

Image data augmentation is typically only applied to the training dataset, and not to the validation or test dataset. This is different from data preparation such as image resizing and pixel scaling.

References:

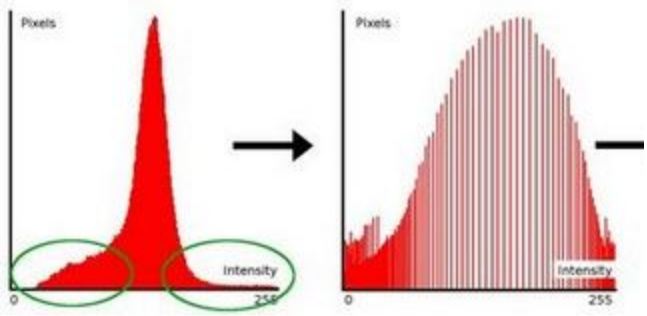
1. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
2. https://iq.opengenus.org/data-augmentation/

1: Histogram Equalization

Method: Histogram Equalization

It is a graphical representation of the intensity distribution of an image. It quantifies the number of pixels for each intensity value considered. It is a method that improves the contrast in an image, in order to stretch out the intensity range.

Histogram Equalization is to stretch out intensities of pixels that are clustered in an image. Take a look at the figure below: The green circles indicate the *underpopulated* intensities. After applying the equalization, we get a histogram like a figure in the center. The resulting image is shown in the picture at right.



Implementation:

1. Loads an image
2. Convert the original image to grayscale
3. Equalize the Histogram by using the OpenCV function.
4. Display the source and equalized images in a window.

Code:

```
# Creating histogram equalized copies
for i in glob.glob(r"D:\DS\Capstone\Train images -2 - Copy- Single Down-Upsize Q2\*"):
    i = i+"\*"
    j=0
    for file in glob.glob(i):
        j+=1
        img=cv2.imread(file,cv2.IMREAD_UNCHANGED)
        img_y_cr_cb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
        y, cr, cb = cv2.split(img_y_cr_cb)
        y_eq = cv2.equalizeHist(y)
        img_y_cr_cb_eq = cv2.merge((y_eq, cr, cb))
        img_rgb_eq = cv2.cvtColor(img_y_cr_cb_eq, cv2.COLOR_YCR_CB2BGR)
        cv2.imwrite(str(file[:-4])+'hist'+str(j)+'.png',img_rgb_eq)
```

References:

1. https://medium.com/@animeshsk3/back-to-basics-part-1-histogram-equalization-in-image-processing-f607f33c5d55

2: PCA

Method: Principal Component Analysis

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional subspace. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variations. In our experiment, we are using PCA to blur the training data by only retaining a percentage of the original variance.

Implementation:

1. Loading modules and Image data.
2. Data Standardization.
3. PCA Transformation.
4. Eigenvalues and Eigenvectors computation.
5. Validation of Principal components.
6. Converting Principal components back to images.
7. Comparison of PC image and RGB image.

References:

1. *https://www.researchgate.net/publication/316652806_Principal_Component_Analysis/link/5b657a4f458515cf1d33a20a/download*
2. *https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4792409/*

3: Gamma adjustment

Method: Gamma- Correction

Gamma correction is also known as the *Power Law Transform*. First, our image pixel intensities must be scaled from the range *[0, 255]* to *[0, 1.0]*. From there, we obtain our output gamma-corrected image by applying the following equation:

O = I ^ (1 / G)

Where *I* is our input image and *G* is our gamma value. The output image *O* is then scaled back to the range *[0, 255]*.

Gamma values ***< 1*** will shift the image towards the darker end of the spectrum while gamma values ***> 1*** will make the image appear lighter. A gamma value of *G=1* will have no effect on the input image.

Implementation:

1. As Python + OpenCV represents images as NumPy arrays we scale the pixel intensities to the range [0,1.0].
2. Applied transform and then scale back to the range [0,255].
3. The NumPy approach involves a division, raising to a power, followed by a multiplication — this tends to be very fast since all these operations are vectorized.

Code:

```
def adjust_gamma(image, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255 for i in np.arange(0, 256)]).astype("uint8")
    return cv2.LUT(image, table).astype("uint8")
```

```
# Changing gamma
for i in glob.glob(r"D:\DS\Capstone\Train images -2 - Copy- Single Down-Upsize Q2\*"):
    i = i+"\*"
    j=0
    for file in glob.glob(i):
        img=cv2.imread(file,cv2.IMREAD_UNCHANGED)
        for m in [0.9,1.1]:
            j+=1
            adjusted = adjust_gamma(img, gamma=m)
            cv2.imwrite(str(i[:-1])+' gamma'+str(j)+'.png',adjusted)
```

References:

1. https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction/
2. https://www.codepool.biz/image-processing-opencv-gamma-correction.html

4: Flipping

An image flip means reversing the rows or columns of pixels in the case of a vertical or horizontal flip respectively.

The flip augmentation is specified by a boolean *horizontal_flip* or *vertical_flip* argument to the *ImageDataGenerator* class constructor.

Code:

```
# Flipping images
for i in glob.glob(r"D:\DS\Capstone\Train images -2 - Copy- Single Down-Upsize Q2\*"):
    i = i+"\*"
    j=0
    for file in glob.glob(i):
        img=cv2.imread(file,cv2.IMREAD_UNCHANGED)
        j+=1
        flipHorizontal = cv2.flip(img, 1)
        cv2.imwrite(str(i[:-1])+' flipped'+str(j)+'.png',flipHorizontal)
```

5: Rotation

A rotation augmentation randomly rotates the image clockwise by a given number of degrees from 0 to 360. The rotation will likely rotate pixels out of the image frame and leave areas of the frame with no pixel data that must be filled in.

Code:

```
# Rotating images
for i in glob.glob(r"D:\DS\Capstone\Blurred training images\*"):
    i = i+"\*"
    j=1
    for file in glob.glob(i):
        img=cv2.imread(file,cv2.IMREAD_UNCHANGED)
        for p in range(1,11,3):
            rotate=iaa.Affine(rotate=(p))
            rotated_image=rotate.augment_image(img)
            cv2.imwrite(str(i[:-1])+' rotated'+str(j)+'.png',rotated_image)
            j+=1
        for q in range(-1,-11,-3):
            rotate=iaa.Affine(rotate=(q))
            rotated_image=rotate.augment_image(img)
            cv2.imwrite(str(i[:-1])+' rotated'+str(j)+'.png',rotated_image)
            j+=1
```

Improvement 3: Training with greyscale.

The reason for differentiating images from any other sort of color image is because it has less information that needs to be presented for each pixel. The ‘gray’ color is one in which the red, green and blue components all have equal intensity in RGB space. Often, the grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white.

Importance of grayscaling -

Dimension reduction: In RGB images there are three color channels and has three dimensions while grayscale images are single-dimensional.
Reduces model complexity: If a training neural architecture on RGB images with the input layer has 300 input nodes, the same neural network will need only 100 input nodes for grayscaled images.
Signal to noise: For many applications of image processing, color information doesn't help us identify important edges or other features. There are exceptions. If there is an edge (a step-change in pixel value) in a hue that is hard to detect in a grayscale image, or if we need to identify objects of known hue (orange fruit in front of green leaves), then color information could be useful. If we don't need color, then we can consider it noise. At first, it's a bit counterintuitive to "think" in grayscale, but you get used to it.

References:

https://stackoverflow.com/questions/12752168/why-we-should-use-gray-scale-for-image-processing

Improvement 4: Training with different images(Low-quality webcam)

Experiment 2:

Method: Training using Super-resolution on Test Images.

Super-resolution is the process of upscaling and or improving the details within an image. We have taken a low-resolution image as an input and the same image is upscaled to a higher resolution, which is the output. The objective is to improve the low-resolution image to be as good (or better) than the target, known as the ground truth, which in this situation is the original image we downscaled into the low-resolution image. To accomplish this a mathematical function takes the low-resolution image that lacks details and hallucinates the details and features onto it. In doing so the function finds detail potentially never recorded by the original camera.

This mathematical function is known as the model and the upscaled image is the model's prediction.

Research Paper:
Deep Learning for Single Image Super-Resolution:

References:
1. https://www.codepool.biz/image-processing-opencv-gamma-correction.html
2. https://cv-tricks.com/deep-learning-2/image-super-resolution-to-enhance-photo
3. https://github.com/idealo/image-super-resolution

Experiment 3:

Method: Resizing of Images.
Image resizing refers to the scaling of images. Scaling comes handy in many image processing as well as machine learning applications. It helps in reducing the number of pixels from an image and that has several advantages e.g. It can reduce the time of training of a neural network as more is the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model.

It also helps in zooming in images. Many times we need to resize the image i.e. either shirk it or scale up to meet the size requirements. OpenCV provides us several interpolation methods for resizing an image.

Here we are using upsizing and downsizing to reduce quality and training in order to blur the training data.

Improvement 1: Downsizing and upsizing single time
Code:

```
#Upsizing image
for i in glob.glob(r"D:\DS\Capstone\Train images -2 - Copy- Single Down-Upsize Q2\*"):
    i = i+"\*"
    j=0
    for file in glob.glob(i):
        j+=1
        img=imread(file)
        if img.shape[1]<200:
            img=resize(img,(img.shape[0]*1.5,img.shape[1]*1.5))
            imsave(file,img_as_ubyte(img))
        else:
            img=resize(img,(img.shape[0]*1.5,img.shape[1]*1.5))
            imsave(file,img_as_ubyte(img))
```

Improvement 2: Downsizing and upsizing 2 times
Code:

```
#Downsizing and upsizing again
for i in glob.glob(r"D:\DS\Capstone\Train images -2 - Copy\*"):
    i = i+"\*"
    j=0
    for file in glob.glob(i):
        j+=1
        img=imread(file)
        if img.shape[1]>1000:
            img=resize(img,(img.shape[0]*0.5,img.shape[1]*0.5))
            img=resize(img,(img.shape[0]*2,img.shape[1]*2))
            imsave(file,img_as_ubyte(img))
        else:
            img=resize(img,(img.shape[0]*0.5,img.shape[1]*0.5))
            img=resize(img,(img.shape[0]*2,img.shape[1]*2))
            imsave(file,img_as_ubyte(img))
```

Experiment 4:

Method: Facenet

In this approach we presented a system, know as FaceNet, that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. Once this space has been produced, tasks such as face recognition, verification and clustering can be easily implemented using standard techniques with FaceNet embeddings as feature vectors.

This method uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer as in previous deep learning approaches.

The FaceNet system is used here to extract high-quality features from faces, called face embeddings, that can then be used to train a face identification system.

***Research Paper**:*
FaceNet: A Unified Embedding for Face Recognition and Clustering
References:
1)https://arxiv.org/abs/1503.03832
2)https://github.com/davidsandberg/facenet

AWS -Amazon Rekognition

Amazon Rekognition makes it easy to add image and video analysis to our applications using proven, highly scalable, deep learning technology that requires no machine learning expertise to use. With Amazon Rekognition, we can identify objects, people, text, scenes, and activities in images and videos, as well as detect any inappropriate content. Amazon Rekognition also provides highly accurate facial analysis and facial search capabilities that we can use to detect, analyze, and compare faces for a wide variety of user verification, people counting, and public safety use cases.

How it works: Amazon Rekognition has multiple options for face detection, recognition, pre-processing, etc.. We are focusing only on the **comparison feature.**

In Amazon Rekognition when we specify a **Reference face** (source) and a **Comparison faces** (target) image, Rekognition compares the largest face in the source image (that is, the reference face) with up to 100 faces detected in the target image (that is, the comparison faces), and then finds how closely the face in the source matches the faces in the target image. The similarity score for each comparison is displayed in the **Results** pane.

If the target image contains multiple faces, Rekognition matches the face in the source image with up to 100 faces detected in the target image and then assigns a similarity score to each match. If the source image contains multiple faces, the service detects the largest face in the source image and uses it to compare with each face detected in the target image.

Metrics Used: To compare a face in the **source image** with each face in the **target image**, Amazon rekognization uses the metrics known as similarity score.

We pass the input and target images as base64-encoded image bytes or as references to images in an Amazon S3 bucket. The image must be formatted as a PNG or JPEG file. In response, the operation returns an array of face matches ordered by similarity score in descending order. For each face match, the response provides a bounding box of the face, facial landmarks, pose details (pitch, roll, and yaw), quality (brightness and sharpness), and confidence value (indicating the

level of confidence that the bounding box contains a face). The response also provides a **similarity score**, which indicates how closely the faces match.

Note: *If the source image contains multiple faces, the service detects the largest face and compares it with each face detected in the target image.*

By default, only faces with a similarity score of greater than or equal to 80% are returned in the response.

References:
https://docs.aws.amazon.com/rekognition/latest/dg/compare-faces-console.html
https://docs.aws.amazon.com/rekognition/latest/dg/API_CompareFaces.html

Experimental Results and Accuracies

Experiment 1- Resnet

•Train- Basic images (500 images)

•Test- mobile captured images

•Accuracy- 3/26

Experiment 2- Resnet

•Train- Augmented images (2.1 lakh images)

•Test- mobile captured images

•Accuracy- 10/26

Experiment 3- Resnet

•Train- Augmented images (2.1 lakh images)

•Test- Super res mobile captured images 1

•Accuracy- 9/26

Experiment 4- Resnet

•Train- Augmented images (2.1 lakh images)

•Test- Super res mobile captured images 2

•Accuracy- 8/26

Experiment 5- Resnet

•Train- Augmented images (2.1 lakh images)

•Test- Super res mobile captured images 3

•Accuracy- 8/26

Experiment 6- Resnet

•Train- Augmented images (2.1 lakh images)

•Test- Super res mobile captured images 4

•Accuracy- 7/26

Experiment 7- Resnet

•Train- Greyscale Augmented images- 96k images

•Test- Greyscale mobile captured images

•Accuracy- 7/26

Experiment 8- Resnet

•Train- Augmented images- 2.1 lakh images- input 100x100 images

•Test- Super res mobile captured images 1

•Accuracy- 11/26

Experiment 9- Resnet

•Train- Double Downsize upsize images- 6k images

•Test- Mobile captured images

•Accuracy- 3/26

Experiment 10- Resnet

•Train- Single Downsize upsize images- 6k images

•Test- Mobile captured images

•Accuracy- 4/26 to 6/26

Experiment 11- Pretrained Facenet

•Train- Class photos-1 per class member

•Test- CCTV images/Mobile captured

•Accuracy- 29/34 & 33/34

Experiment 12- Resnet

•Train- Webcam low res- 550 images of 6 people

•Test- CCTV+mobile images of class

•Accuracy- 6/9

Experiment 13- AWS Rekognition

•Train- Class photos-1 per class member

•Test- CCTV images/Mobile captured

•Accuracy- 32/34 & 34/34

Result

After implementing Amazon Rekognization's comparison features we found that performance is so good that even though our images are from the different source it can identify most correctly with a similarity score of 90%.

1. We also came to know that faces whose orientation is not towards the camera(CCTV) are being identified.
2. Moreover, it can easily handle a face that actually does not belong to any existing class as with similarity metrics, there is no force that all faces should be matched. (In resnet that external face will be forcibly classified to one of the existing training data class.)

3. Out of 38 faces, Amazon Rekognization identified everyone properly except 2 faces (Nikhil and Soundarya), even those 2 were properly recognized if we take top2 result accuracy.
4. On average out of 38 faces present, it can classify 36 faces correctly. Accuracy 36/38.
5. The cases in which it can not decide we have included some checks and even after that if it's inconclusive then we give that as N/A in attendance which can be manually reviewed.

Deployment

•Rest API –

–REpresentational State Transfer (REST) API is a way of accessing web services in a simple and flexible way without having any processing.

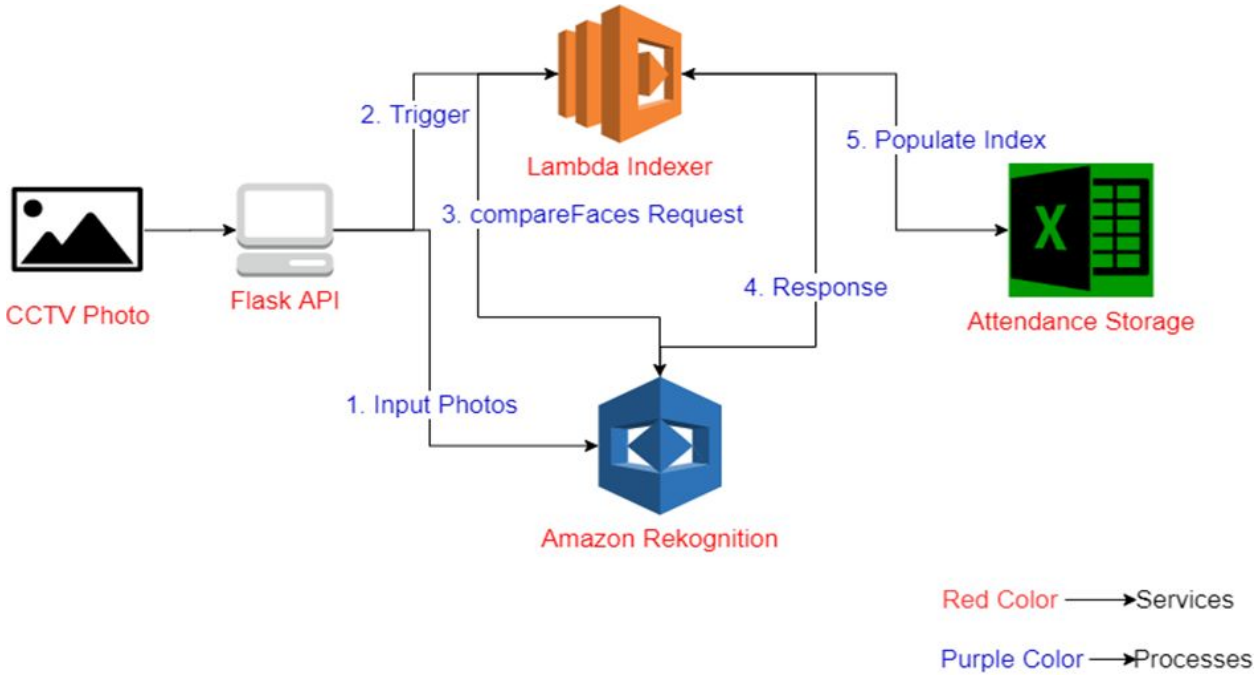
•Flask –

–It is a Python micro-framework that is used to develop web applications and is based on the Werkzeug WSGI toolkit and Jinja2 template engine.

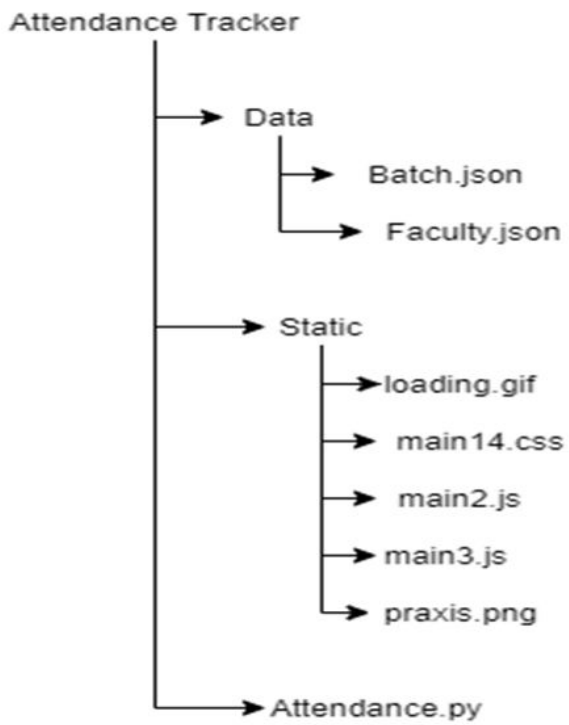
•AWS Rekognition -

–Amazon Rekognition is based on highly scalable, deep learning technology developed by Amazon's computer vision scientists to analyze billions of images and videos daily.

Architecture



Flask Application skeleton [Main]



New Batch Folder Structure