

PROJECT REPORT ON

Face Recognition-based Attendance System

Submitted to

Department of Computer Applications

in partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

Batch (2022-2024)

Submitted by

Priyanka Yadav

1102545

Under the Guidance of

Dr. Varsha Mittal



GRAPHIC ERA DEEMED TO BE UNIVERSITY DEHRADUN

July -2023



Graphic Era

Deemed to be University

CANDIDATE'S DECLARATION

I hereby certify that the work presented in this project report entitled “_____” in partial fulfilment of the requirements for the award of the degree of Master of Computer Applications is a bona fide work carried out by me during the period of March 2023 to June 2023 under the supervision of _____, Department of Computer Application, Graphic Era Deemed to be University, Dehradun, India.

This work has not been submitted elsewhere for the award of a degree/diploma/certificate.

Name and Signature of Candidate

This is to certify that the mentioned statement in the candidate's declaration is correct to the best of my knowledge.

Date: _____

Name and Signature of Guide

Signature of Supervisor

Signature of External Examiner

CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled _____

submitted to **Graphic Era University, Dehradun** in partial fulfilment of the requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS (MCA)**, is an authentic and original work carried out by Mr. / Ms. _____ with enrolment number _____ under my supervision and guidance.

The matter embodied in this project is genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfilment of the requirements of any course of study.

.....

Signature of the Student:

Date :

Enrolment No.: _____

Signature of the Guide

Date:

Name of the Student:

Name, Designation Address of the Guide:

Special Note:

Acknowledgement

I would like to express my deepest gratitude and appreciation to all the individuals and organizations who have contributed to the successful completion of this project report on "Face Recognition-based Attendance System." As a MCA student, this project has been an invaluable opportunity for me to apply the knowledge and skills acquired during my academic journey.

First and foremost, I would like to extend my heartfelt thanks to my project guide Dr. Varsha Mittal, for his/her continuous guidance, valuable support and unwavering encouragement throughout the entire duration of this project. His/her expertise, patience, and insightful suggestions have been instrumental in shaping this report and enhancing the overall quality of the project.

Candidate Name and Signature

Table of Contents i

CHAPTER 1 INTRODUCTION	7-10
1.1 Project Introduction	7
1.2 Motivation	8
1.3 Objective	8-9
1.4 Scope	9
1.5 SRS	9-10
 CHAPTER 2 SYSTEM DESIGN	 11-17
2.1 System Flow	11
2.2 Flow Chart	12
2.3 Algorithms	13-17
 CHAPTER 3 LITERATURE REVIEW	 19 – 41
3.1 Attendance System	19
3.2 Digital Image Processing	20
3.3 Image Representation in a Digital Computer	20
3.4 Steps in Digital Image Processing	21
3.5 Face Detection	22
3.6 Difference between Recognition and Detection	22
3.6 Face Recognition	22-34

CHAPTER 4 CODE IMPLEMENTATION	35-46
4.1 Libraries	35
4.2 Data Retrieval	36
4.3 Names for Attendance	37
4.4 Finding Encodings	38-39
4.5 Mark Attendance	39-40
4.6 WebCam Access	40-41
4.7 Resize	41
4.8 Finding Face locations	42
4.9 Recognizing	42-43
4.10 Matching	44-45
4.11 WebCam	45
4.12 Results	46-47
CONCLUSION	48
REFERENCES	49

INTRODUCTION

1.1 Project Introduction:

The “**Face Recognition-based Attendance System**” is an attendance management system developed using Python and OpenCV. This system aims to automate the attendance process in educational institutions or organizations by leveraging the power of computer vision and machine learning.

The system operates by capturing live video feed from a webcam or any other suitable camera source. It then processes each frame using OpenCV, a popular computer vision library in Python, to detect and recognize human faces. The face detection algorithm identifies the presence of faces within the frame, while the face recognition algorithm matches the detected faces with pre-registered faces in the system's database.

The development of the “**Face Recognition-based Attendance System**” involves the following key steps:

1. **Face Data Collection:** Initially, the system requires a collection of face data to create a face database for recognition. This involves capturing images of individuals who will be part of the attendance system. These images serve as a reference for future attendance verification.
2. **Face Detection:** OpenCV's pre-trained face detection model is utilized to locate and isolate faces within the video frames. The face detection algorithm employs a combination of Haar cascades or deep learning-based models to accurately detect facial regions.
3. **Face Recognition:** The detected faces are matched against the pre-registered faces in the system's database using a face recognition algorithm. Commonly used approaches include eigenfaces, Fisherfaces, or deep learning-based models such as Convolutional Neural Networks (CNNs). The recognition algorithm calculates a similarity score to determine the identity of the detected face.
4. **Attendance Recording:** Once a recognized face is matched with a registered identity, the system marks the attendance for that individual. It records the date, time, and the name of the person, which can be stored in a local database or an external system.
5. **Integration and Scalability:** The system can be integrated with existing attendance management systems or can serve as a standalone solution. It offers scalability by allowing the addition or removal of individuals from the face database, ensuring up-to-date attendance tracking.

1.2 Motivation:

Ensuring attendance records are accurately maintained is of paramount importance for educational organizations, benefiting both teachers and students alike. However, the traditional process of manually calling out names or roll numbers for attendance is not only time-consuming but also physically demanding. To overcome these challenges, the implementation of an automatic attendance system becomes crucial.

Several automatic attendance systems, such as biometric techniques and RFID systems, are currently utilized by many institutions. While these advancements represent significant improvements over traditional methods, they still fall short in meeting time constraints. In such systems, students often find themselves waiting in queues to register their attendance, leading to unnecessary delays.

To address these issues, this project introduces an innovative involuntary attendance marking system that seamlessly integrates with the normal teaching procedure. This system is equally applicable during examination sessions or other teaching activities where attendance tracking is essential. By eliminating the need for classical student identification methods like verbal calls or manual card checks, potential disruptions to the ongoing teaching process are eliminated. Furthermore, this approach minimizes the stress students may experience during examination sessions. Notably, students can conveniently register themselves in the system's database through a user-friendly interface, simplifying the enrollment process.

1.3 Objective:

The Attendance System provides an efficient and accurate means of automating attendance management. By leveraging the power of Python, OpenCV, and face recognition algorithms, the system streamlines the process, reduces errors, and eliminates the need for manual attendance tracking. Here are some main objectives of the attendance system:

1. **Automate Attendance Management:** The primary objective of the system is to automate the attendance management process in educational institutions or organizations. By using face recognition technology, the system eliminates the need for manual attendance tracking, reducing administrative burden and potential errors.
2. **Enhance Accuracy and Efficiency:** The system aims to improve the accuracy and efficiency of attendance recording. Face recognition algorithms can accurately match detected faces with pre-registered identities, ensuring reliable attendance tracking without the possibility of proxy attendance.

3. **Provide Real-time Attendance Tracking:** The system enables real-time attendance tracking by processing live video feed from a camera source. It swiftly detects and recognizes faces, promptly marking attendance records, and allowing administrators to access up-to-date attendance information.

4. **Cost-effectiveness:** Developing the Attendance System using Python and OpenCV aims to provide a cost-effective solution compared to commercial alternatives. Open-source libraries and frameworks contribute to minimizing the overall cost while maintaining functionality and performance.

By accomplishing these objectives, the Attendance System offers an efficient, accurate, and user-friendly solution for attendance management in educational institutions and organizations.

1.4 Scope:

This module utilizes advanced facial recognition technology to identify and analyze captured images of faces within a given file. Its primary purpose is to automatically mark the attendance of students based on their recognized faces. The module further facilitates the storage of the attendance records in a secure database, enabling future analysis and evaluation.

1.5 SRS:

1.5.1 Hardware Requirements:

1. **Camera:** A webcam or any other suitable camera capable of capturing video feed is required for the Face Recognition-based Attendance System. The camera should provide adequate resolution and frame rate to ensure clear and smooth video capture.
2. **Computer System:** A computer system capable of running Python and supporting OpenCV is necessary. A standard modern computer with a decent processor, RAM, and storage capacity should be sufficient for most applications.
 - **Processor:** A modern processor, such as an Intel Core i5 or equivalent, is sufficient for running the code effectively.
 - **RAM:** A minimum of 4GB of RAM is recommended. However, larger datasets or more complex computations may benefit from having more RAM.
 - **Storage:** Adequate storage capacity to store the dataset and any additional files generated during the project.

1.5.2 Software Requirements:

1. **Python:** The Face Recognition-based Attendance System is developed using the Python programming language. Ensure that Python is installed on the computer system. Install the required libraries using pip or conda package managers. Here are the libraries needed for this project:

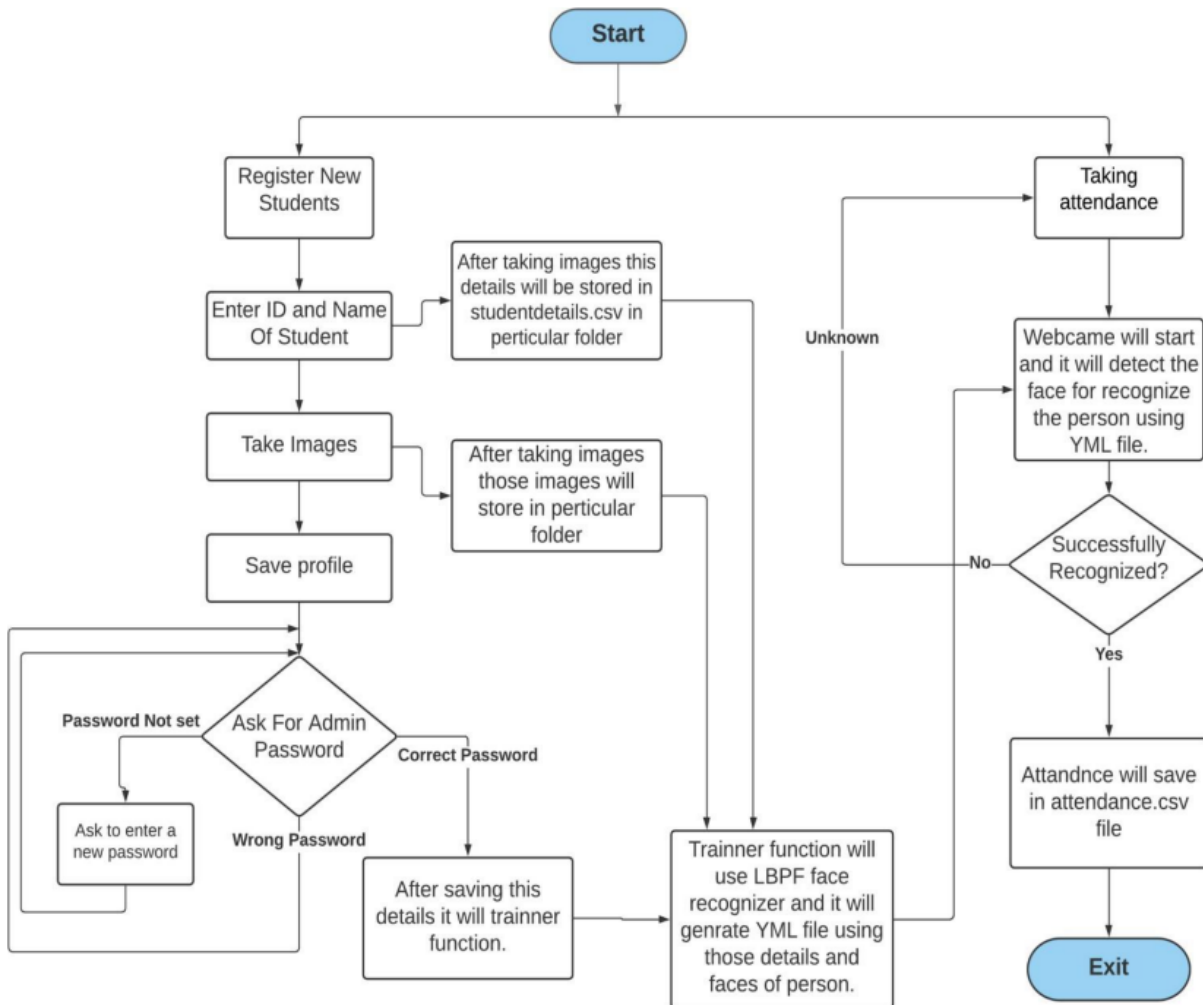
- **CMake :** CMake is a cross-platform build tool used to automate the building and packaging of software projects. It generates build files that can be used on different operating systems and build environments. It simplifies the process of building and distributing software across platforms. Download and install CMake from the official website.
- **NumPy :** NumPy is a Python library for efficient numerical computations and array manipulation. Install using “pip install numpy”
- **Dlib :** Dlib is a library for machine learning, image processing, and computer vision. It requires CMake and some additional dependencies to be installed before installing the Python bindings. Follow these steps to install dlib:
 - Install CMake as mentioned in step 1.
 - Install Visual Studio (Community Edition or any other edition) with the "Desktop development with C++" workload selected.
 - Open the Command Prompt or PowerShell as an administrator.
 - Install using “pip install dlib”
- **face-recognition :** face-recognition is a simple library for face recognition and manipulation. Install using “pip install face-recognition”
- **OpenCV-Python :** OpenCV-Python is a library for computer vision and image processing. Install using “pip install opencv-python”

2. **OpenCV:** OpenCV (Open-Source Computer Vision Library) is a popular open-source computer vision and machine learning library. Install the appropriate version of OpenCV for Python on the system.

3. **Integrated Development Environment (IDE):** An IDE such as PyCharm, Visual Studio Code, or Jupyter Notebook can be used for coding and development.

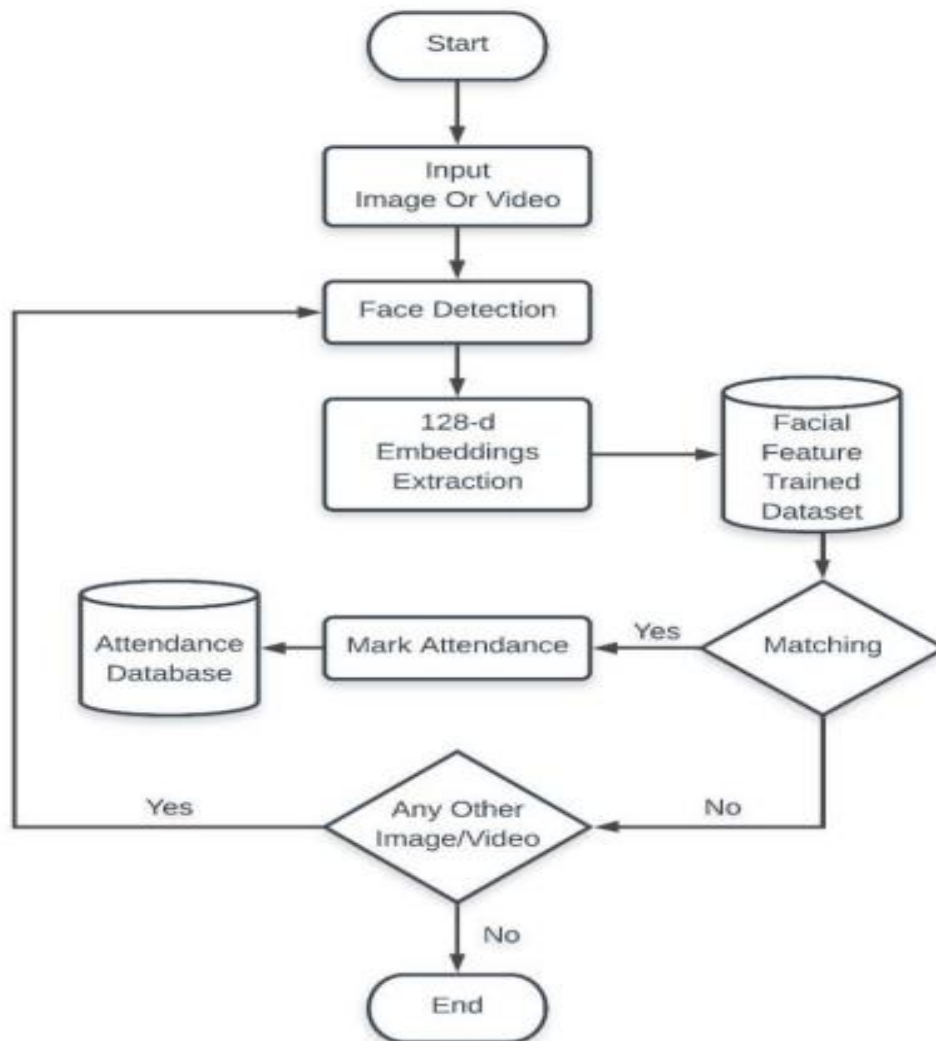
SYSTEM DESIGN

2.1 System Chart:



(System chart)

2.2 Flow Chart:

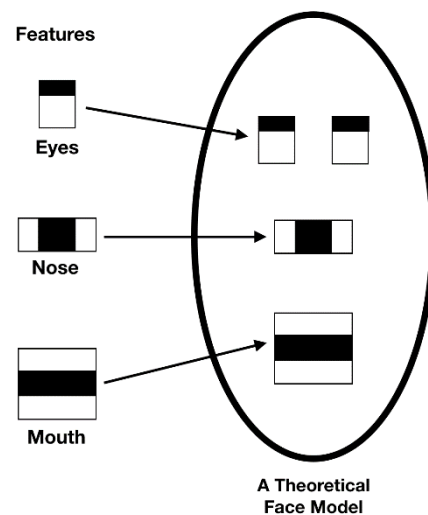


(Flow chart)

2.3 Algorithms:

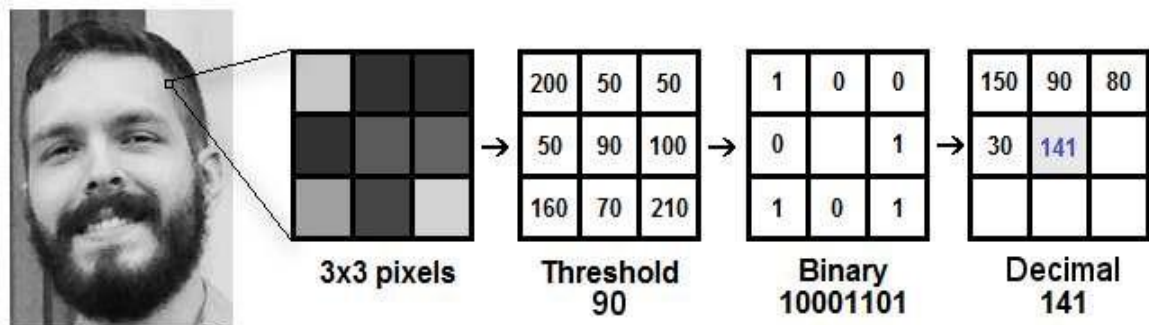
1. Haar cascade Algorithm:

It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images (where positive images are those where the object to be detected is present, negative are those where it is not). It is then used to detect objects in other images. Luckily, OpenCV offers pre-trained Haar cascade algorithms, organized into categories (faces, eyes and so forth), depending on the images they have been trained on.



LBPH Algorithm:

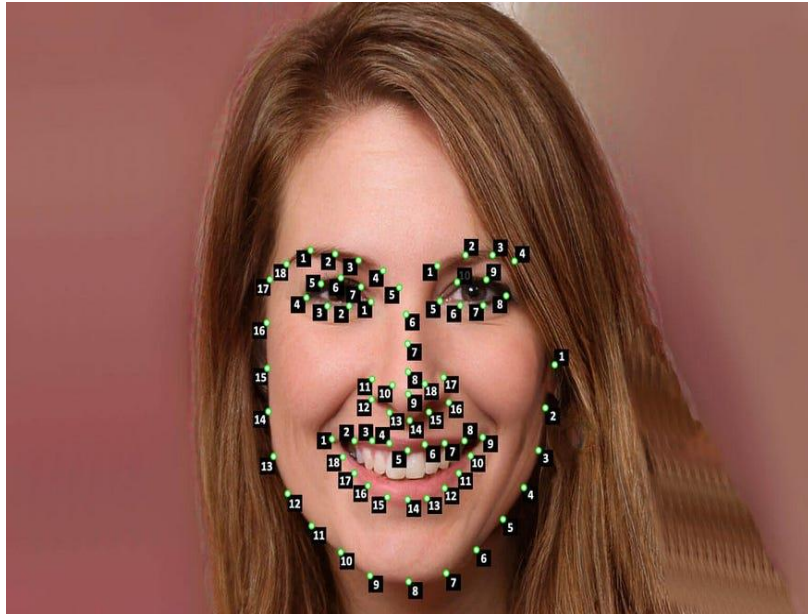
Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. It has further been determined that when LBP is combined with histograms of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.



2. Face landmark estimation algorithm: Face landmark estimation is an algorithmic process that involves detecting and localizing key facial features in an image or video, such as the position of the eyes, nose, mouth, and other facial landmarks. The purpose of this algorithm is to accurately identify and track these landmarks for various applications, including facial recognition, emotion analysis, virtual reality, and augmented reality.

The algorithm typically follows a set of steps to estimate facial landmarks:

- **Face Detection:** Initially, the algorithm identifies and localizes the face(s) in the input image or video frame. This step is crucial because it provides a region of interest within which the subsequent landmark estimation will be performed.
- **Landmark Localization:** Once the face is detected, the algorithm analyzes the facial region in more detail to determine the positions of specific landmarks. Commonly detected landmarks include the corners of the eyes, the tip of the nose, the corners of the mouth, and other characteristic points on the face.
- **Feature Extraction:** After identifying the facial landmarks, the algorithm extracts relevant features from these points. These features may include the distances between landmarks, the angles formed by specific points, or even the shape and curvature of the face. These features can then be used for further analysis or applications.
- **Machine Learning or Deep Learning:** Face landmark estimation algorithms often employ machine learning or deep learning techniques. These algorithms are trained on large datasets of labeled facial images, where the ground truth landmarks are manually annotated. The training process helps the algorithm learn the relationships between different facial features and their corresponding landmark positions.
- **Tracing and Refinement:** In real-time applications, it is essential to track the facial landmarks across consecutive frames or video sequences. Tracking algorithms can utilize motion estimation techniques or optical flow to estimate the new positions of the landmarks in each frame. Additionally, refinement techniques may be employed to improve the accuracy of the landmark estimation over time.



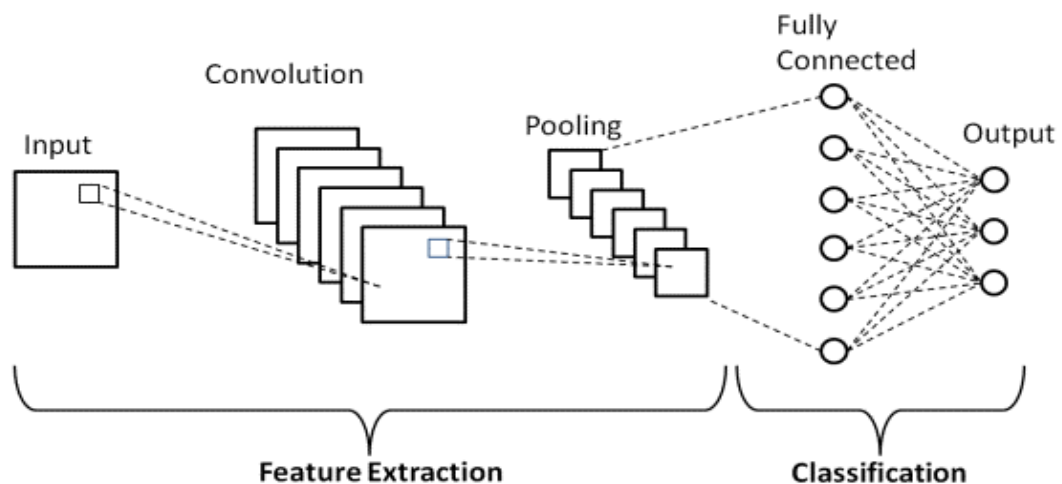
3. **Convolutional Neural Network (CNN):** A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing task. It is particularly effective in tasks related to computer vision, including image classification, object detection, and image segmentation.

The architecture of a CNN is composed of multiple layers, typically including convolutional layers, pooling layers, and fully connected layers. These layers are stacked together to form a deep network that can learn hierarchical representations of the input data.

- Convolutional layers are responsible for extracting local features by convolving filters or kernels across the input image. Each filter captures different visual patterns, such as edges, textures, or shapes. The output of a convolutional layer is called a feature map, which highlights the presence of specific features in the input.
- Pooling layers are used to reduce the spatial dimensions of the feature maps while preserving the most important information. The pooling operation, such as max pooling, downsamples the feature maps by selecting the maximum value within a defined region.
- Fully connected layers are typically placed at the end of the network and are responsible for making the final predictions based on the extracted features. These

layers connect every neuron from the previous layer to every neuron in the current layer, enabling the network to learn complex combinations of features.

One of the significant advantages of CNNs is their ability to automatically learn hierarchical representations of the input data. Through a process called backpropagation, the network adjusts its internal parameters (weights and biases) based on the difference between the predicted output and the ground truth labels. This training process allows the network to optimize its performance over time.



- 4. SVM Classifier:** A Support vector Machine (SVM) classifier is a supervised machine learning algorithm that is primarily used for binary classification tasks, although it can be extended to handle multi-class problems as well. SVMs are known for their effectiveness in separating data into distinct classes by finding an optimal hyperplane that maximally separates the data points of different classes.

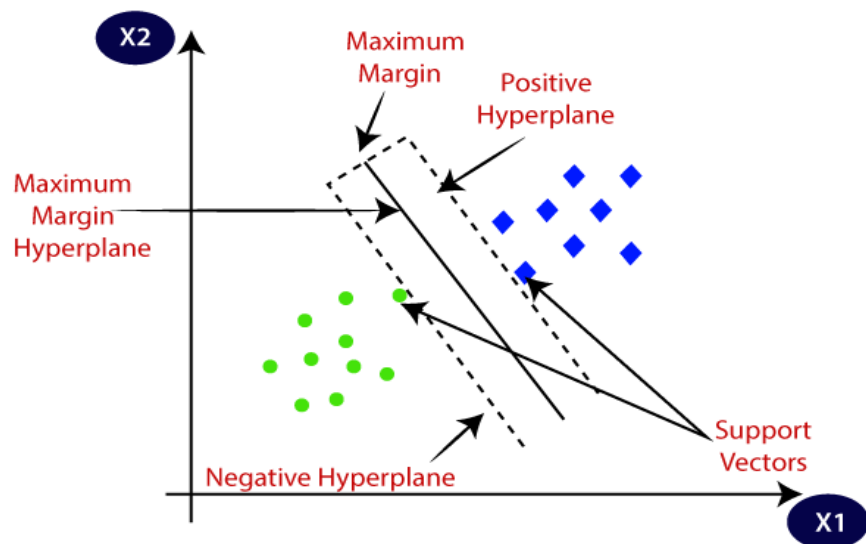
The basic principle of an SVM is to map input data points into a higher-dimensional feature space and then find the hyperplane that best separates the data points of different classes. The hyperplane is chosen such that the margin, which is the distance between the hyperplane and the nearest data points of each class (called support vectors), is maximized.

SVMs can handle linearly separable data by using a linear kernel. However, they can also handle non-linearly separable data by employing various kernel functions, such as polynomial kernels or radial basis function (RBF) kernels. These kernels transform the data into a higher-dimensional space, allowing for more complex decision boundaries.

During the training phase, the SVM algorithm aims to find the optimal hyperplane by solving an optimization problem. The objective is to minimize the classification error while maximizing the margin. This optimization problem can be formulated as a quadratic programming (QP) problem or solved using optimization techniques.

Once the SVM classifier is trained, it can be used to predict the class labels of new, unseen data points by evaluating which side of the hyperplane they fall on. The decision boundary is determined by the support vectors, which are the data points that lie closest to the hyperplane.

SVMs have several advantages, including their ability to handle high-dimensional data, their robustness to overfitting, and their effectiveness in handling small training datasets. However, SVMs can be computationally expensive, especially with large datasets, and they may require careful selection of hyperparameters, such as the regularization parameter (C) and the kernel parameters.



5. Modules:

OpenCV Library:

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

NumPy package:

NumPy is a Python package which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc.

CMake:

CMake is an open-source cross-platform build system that helps manage the build process of software projects. It allows developers to describe the build requirements and dependencies of their project using a CMakeLists.txt file, which provides instructions on how to compile, link, and install the software. CMake generates platform-specific build files (e.g., Makefiles, Visual Studio projects) based on the target platform and build environment, enabling developers to easily build their projects on different operating systems and with various compilers. It simplifies the build configuration process and provides a standardized approach to building and distributing software across multiple platforms.

Dlib:

Dlib is an open-source software library primarily used for developing applications in the field of computer vision and machine learning. It provides a wide range of algorithms and tools that facilitate various tasks, such as face detection, facial landmark detection, object tracking, image classification, and clustering. dlib is written in C++ and is known for its efficiency and high-quality implementations. It offers a simple and intuitive interface, making it accessible for both researchers and developers to integrate computer vision capabilities into their projects.

Time Module:

Python has a module named time to handle time related task. To use functions defined in the module, we need to import the module first.

Date Time Module:

A date in python is not a date type of its own, but we can import a module named date time work with dates as a date objects.

LITREATURE REVIEW

3.1 Attendance System:

Arun Katara et al. (2017) mentioned disadvantages of RFID (Radio Frequency Identification) card system, fingerprint system and iris recognition system. RFID card system is implemented due to its simplicity. However, the user tends to help their friends to check in as long as they have their friend's ID card. The fingerprint system is indeed effective but not efficient because it takes time for the verification process so the user has to line up and perform the verification one by one. However for face recognition, the human face is always exposed and contain less information compared to iris. Iris recognition system which contains more detail might invade the privacy of the user. Voice recognition is available, but it is less accurate compared to other methods. Hence, face recognition system is suggested to be implemented in the student attendance system.

System Type	Advantage	Disadvantages
RFID card system	Simple	Fraudulent usage
Fingerprint system	Accurate	Time-consuming
Voice recognition system		Less accurate compared to Others
Iris recognition system	Accurate	Privacy Invasion

Table 2.1: Advantages & Disadvantages of Different Biometric System

3.2 Digital Image Processing:

Digital Image Processing is the processing of images which are digital in nature by a digital computer. Digital image processing techniques are motivated by three major applications mainly:

- 3.2.1 Improvement of pictorial information for human perception
- 3.2.2 Image processing for autonomous machine application
- 3.2.3 Efficient storage and transmission

3.3 Image Representation in a Digital Computer:

An image is a 2-Dimensional light intensity function

$$f(x,y) = r(x,y) \times i(x,y) \text{ -(2.0)}$$

Where, $r(x, y)$ is the reflectivity of the surface of the corresponding image point. $i(x,y)$ Represents the intensity of the incident light. A digital image $f(x, y)$ is discretized both in spatial co-ordinates by grids and in brightness by quantization. Effectively, the image can be represented as a matrix whose row, column indices specify a point in the image and the element value identifies gray level value at that point. These elements are referred to as pixels or pels.

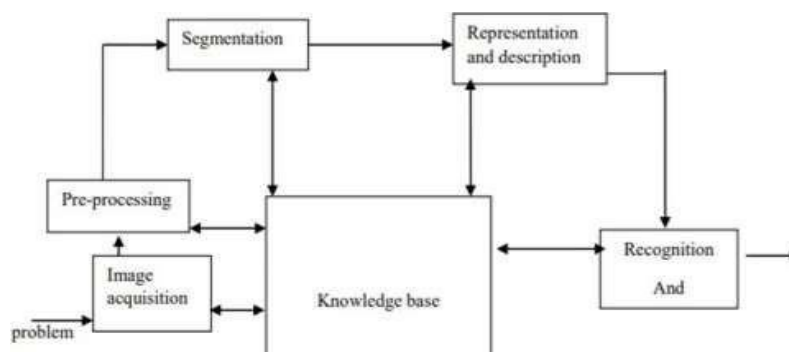
Typically following image processing applications, the image size which is used is 256×256 , elements, 640×480 pels or 1024×1024 pixels. Quantization of these matrix pixels is done at 8 bits for black and white images and 24 bits for colored images (because of the three color planes Red, Green and Blue each at 8 bits)

3.4 Steps in Digital Image Processing:

Digital image processing involves the following basic tasks:

- Image Acquisition - An imaging sensor and the capability to digitize the signal produced by the sensor.
- Preprocessing – Enhances the image quality, filtering, contrast enhancement etc.
- Segmentation – Partitions an input image into constituent parts of objects.
- Description/feature Selection – extracts the description of image objects suitable for further computer processing.
- Recognition and Interpretation – Assigning a label to the object based on the information provided by its descriptor. Interpretation assigns meaning to a set of labelled objects.

Knowledge Base – This helps for efficient processing as well as inter module cooperation.



3.5 Face Detection:

Face detection is the process of identifying and locating all the present faces in a single image or video regardless of their position, scale, orientation, age and expression. Furthermore, the detection should be irrespective of extraneous illumination conditions and the image and video content.

3.5 Difference between Face Detection and Face Recognition:

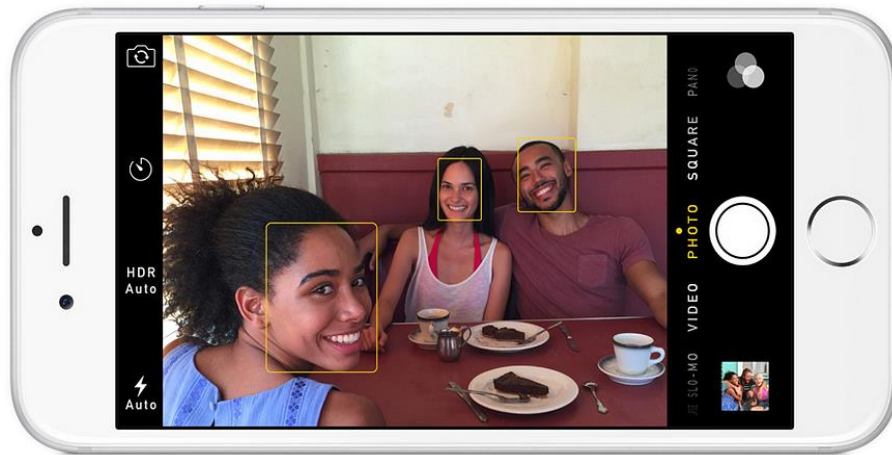
Face detection answers the question, Where is the face? It identifies an object as a “face” and locates it in the input image. Face Recognition on the other hand answers the question who is this? Or whose face is it? It decides if the detected face is someone. It can therefore be seen that face detection's output (the detected face) is the input to the face recognizer and the face Recognition's output is the final decision i.e. face known or face unknown.

3.7 Face Recognition:

The detected faces are matched against the pre-registered faces in the system's database using a face recognition algorithm. Commonly used approaches include eigenfaces, Fisherfaces, or deep learning-based models such as Convolutional Neural Networks (CNNs). The recognition algorithm calculates a similarity score to determine the identity of the detected face.

Step 1: Finding all the Faces:

OpenCV's pre-trained face detection model is utilized to locate and isolate faces within the video frames. The face detection algorithm employs a combination of Haar cascades or deep learning-based models to accurately detect facial regions.

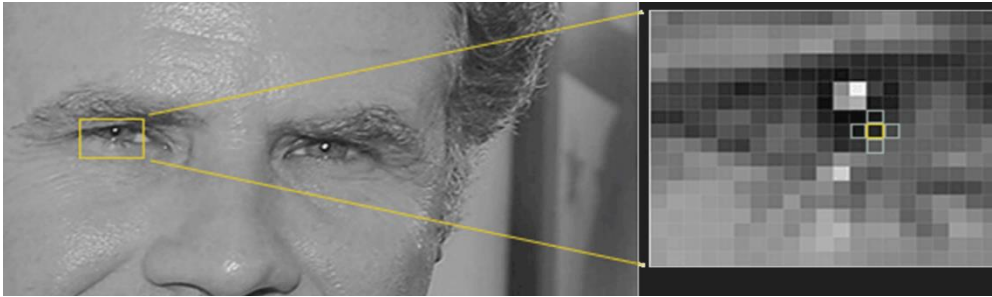


Face detection is a great feature for cameras. When the camera can automatically pick out faces, it can make sure that all the faces are in focus before it takes the picture. In this project it is used for a different purpose that is finding the areas of the image we want to pass on to the next step.

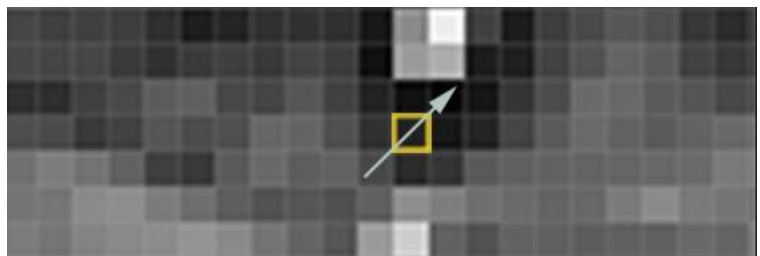
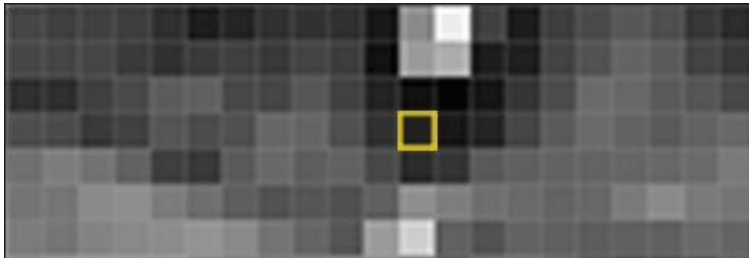
Face detection went mainstream in the early 2000's when Paul Viola and Michael Jones invented a way to detect faces that was fast enough to run on cheap cameras. However, there is a more reliable solution. The approach for this project is Histogram of Oriented Gradients (**HOG**) which was invented in 2005. To find faces in an image, first step is to make the picture black and white as coloured images are not required to find faces:



Then it is required to look at every single pixel in the image one at a time. And also the surrounding pixels:

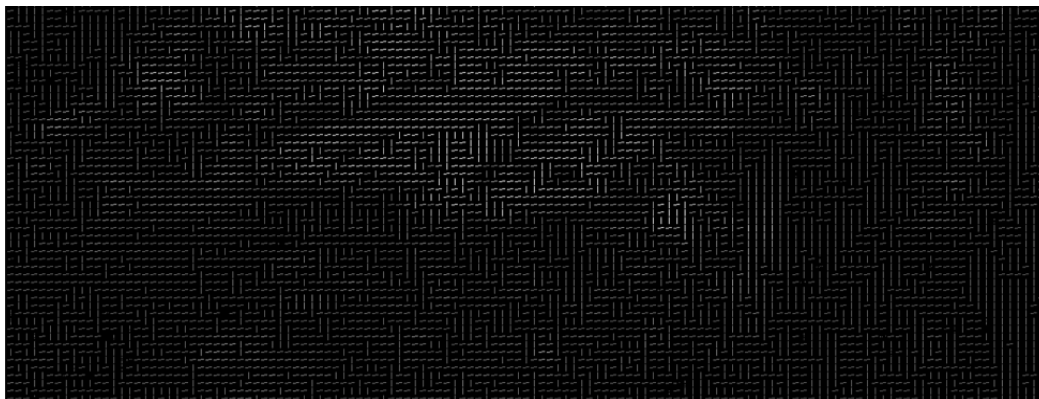
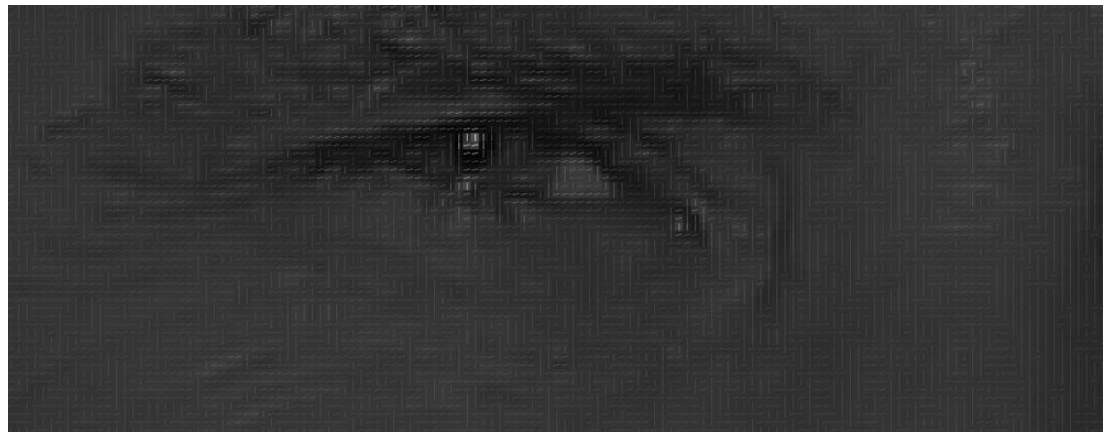


The goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then draw an arrow showing in which direction the image is getting darker:



The above images shows this one pixel and the pixels touching it, the image is getting darker towards the upper right.

By repeating that process for **every single pixel** in the image, every pixel in the image is replaced by an arrow. These arrows are called *gradients* and they show the flow from light to dark across the entire image:



This is to do analysis on pixels as dark image and bright image of the same person will have different pixel values. But if only the direction in which the brightness changes is considered, both dark image and bright image will end up with the same exact representation.

But it is not possible to save the gradient for every single pixel. Its time consuming so it is better to detect basic flow of lightness/darkness at a higher level to find the basic pattern of the image.

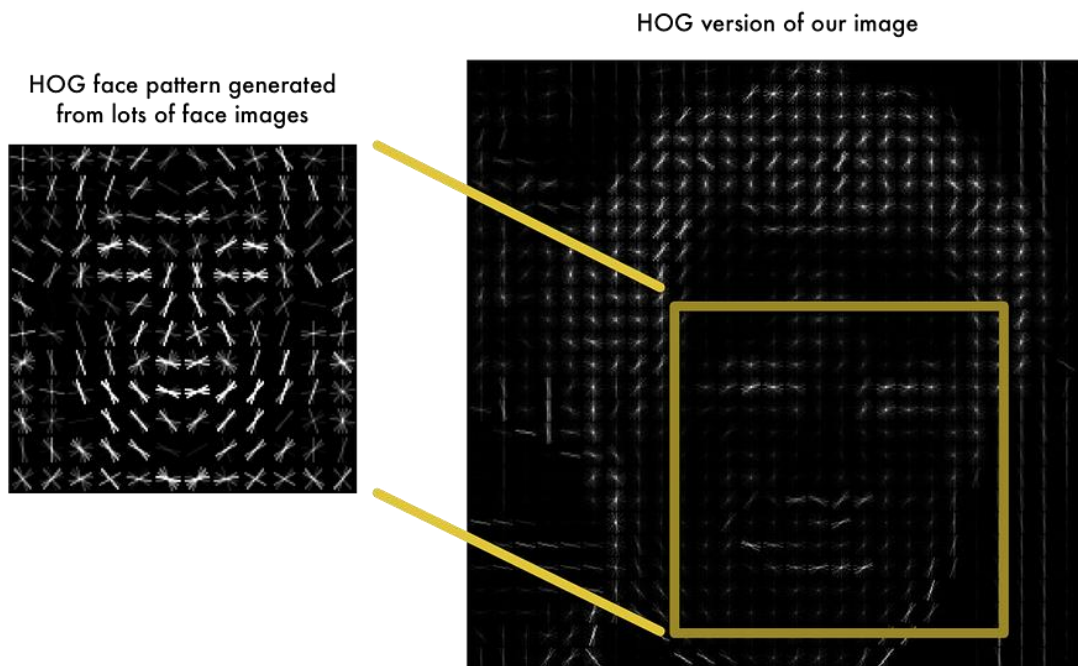
It is done by breaking the image into small squares of 16x16 pixels each. In each square, the number of gradients point in each major direction (how many points to up, right, up-right, etc) needs to be found. Then replace that square in the image with the arrow directions that are more in number.

The result will be the original image will turn into a very simple representation that captures the basic structure of a face in a simple way:

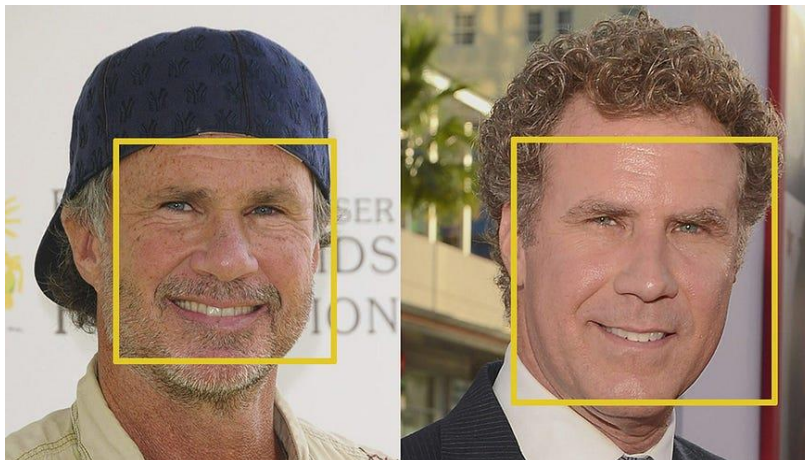


The above images are the original image and HOG representation that captures the major features of the original image. Now this is the major component which is used to recognize the faces by comparing or analyzing it with the images of faces present in the database.

To find faces in this HOG image, analysis is done to find a match by finding the most similar HOG pattern that was extracted from the faces present in the database:

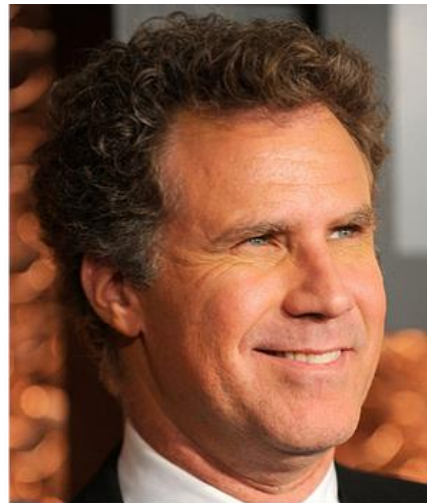
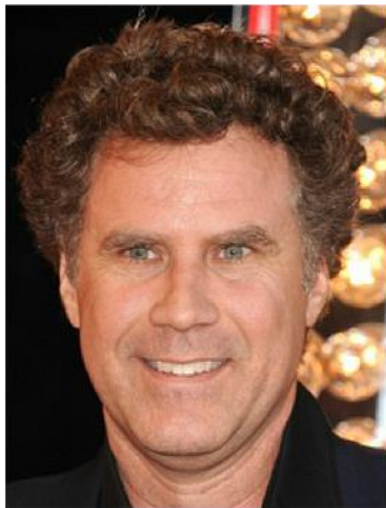


Using this technique, the face can be recognized in any type of image:



Step 2: Posing and Projecting Faces:

By achieving the first step the module will be able to recognize the faces if the images are taken from the same angle. But the problem arises when the images are taken from a different angle as faces turned different directions look different to the module since the HOG representation of the images will not be the same. Hence making the module think they are of different people:

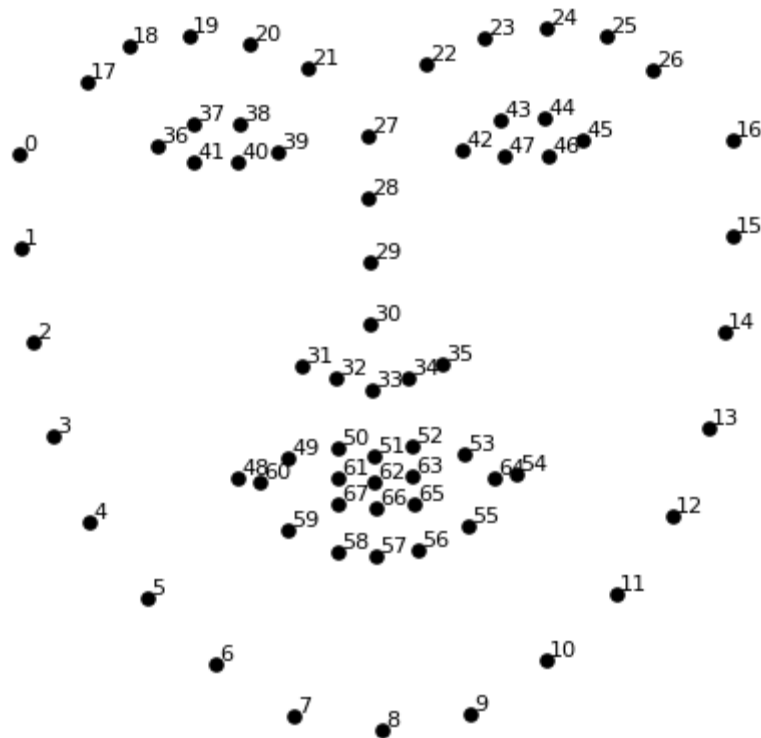


To solve this problem, each image needs to be warp so that the eyes and lips are always in the sample place in the image. This will make it easier to compare faces in the next steps.

The algorithm used to do this is called **face landmark estimation**. Face landmark estimation is an algorithm that detects and locates key facial features, like eyes and mouth, in images or videos. It uses techniques like face detection, landmark localization, feature extraction, and machine learning to accurately identify these landmarks. This information is used in applications such as facial recognition, virtual reality, and augmented reality.

There are different approach to make this possible but this module uses the approach invented in 2014 by Vahid Kazemi and Josephine Sullivan. It is a landmark algorithm called "Face Alignment by Explicit Shape Regression." This algorithm utilizes a cascaded regression framework to iteratively refine the initial face shape estimation by learning the shape variations and regression functions from training data. It has been widely adopted and forms the basis for many subsequent advancements in face landmark estimation.

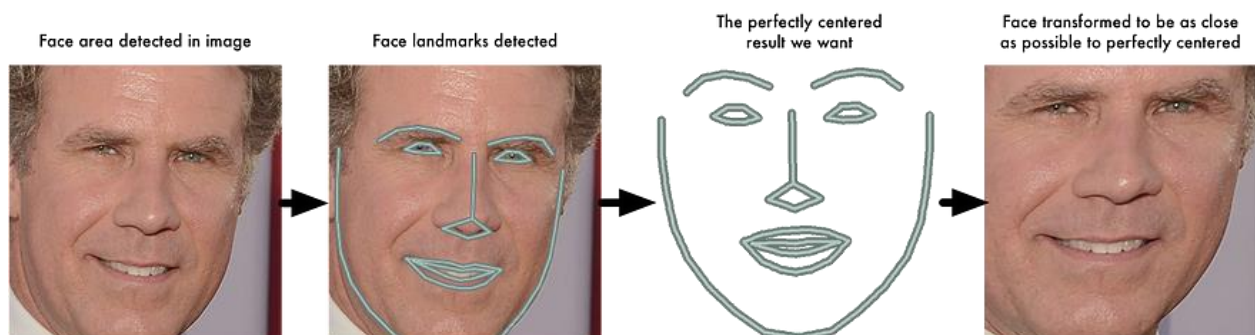
The approach says to find 68 specific points (landmarks) that exist on every face that is the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then a machine learning algorithm is trained to be able to find these 68 specific points on any face:



The result of locating the 68 face landmarks on the test image:



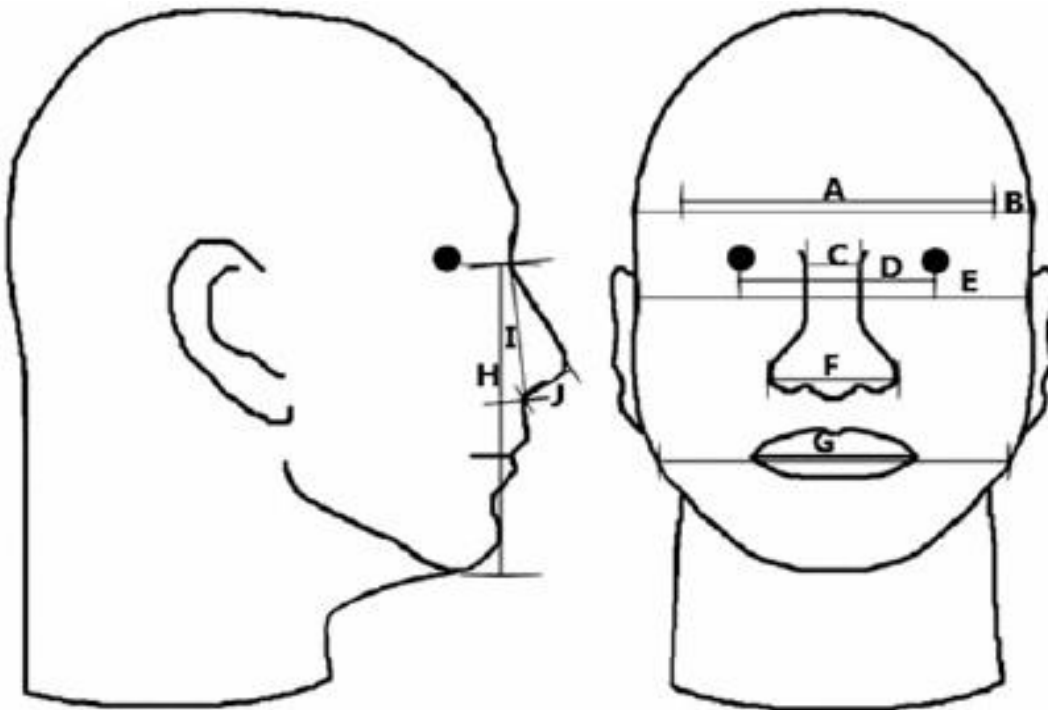
Now that the eyes and mouth are detected, the image needs to be rotated, scaled and sheared so that the eyes and mouth could be aligned in the centre. This is done by using basic image transformations like rotation and scale that preserve parallel lines (affine transformations):



Step 3: Encoding Faces:

By rotating and scaling the image to preserve parallel lines in the previous step, this step will be more accurate. The simplest approach to face recognition is to directly compare the unknown face we found in the previous step with all the images present in the database. But there is a problem with that approach. A site like Facebook with billions of users and a trillion photos cannot possibly loop through every previous-tagged face to compare it to every newly uploaded picture. That would take hours to recognize.

What needs to be done is to find basic measurements from each face. Then find the measurements of captured face the same way and find the face with the closest measurements in the database. For example, measure the size of each ear, the spacing between the eyes, the length of the nose, etc:



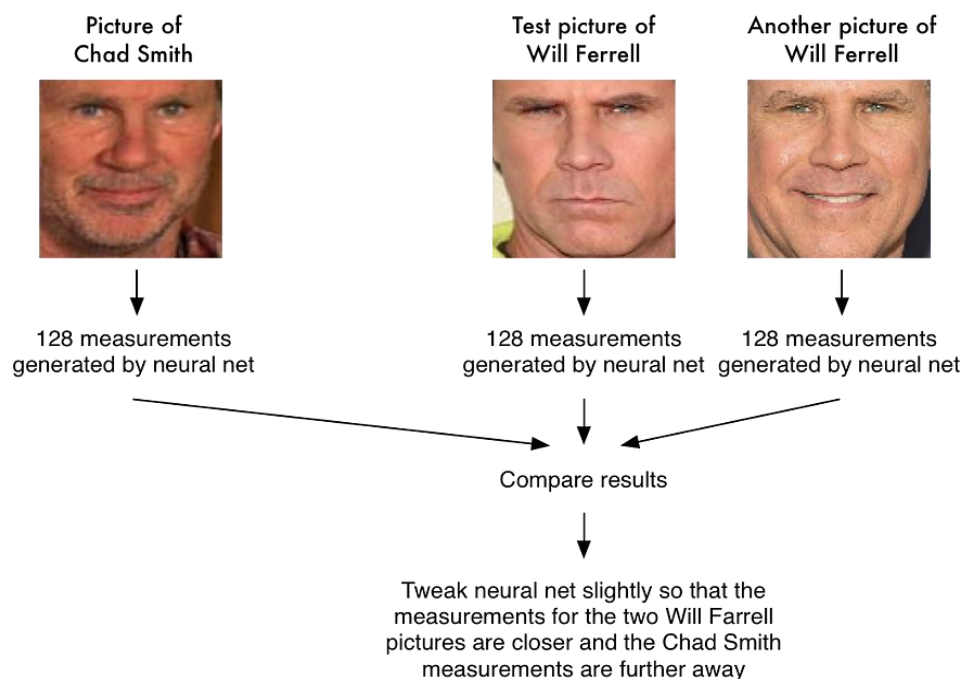
The solution is to train a Deep Convolutional Neural Network (CNN). The goal is to train it to generate 128 measurements for each face.

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:




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. 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. Hence, this step only needs to be done once. OpenFace already did this and they published several trained networks which can be directly used.

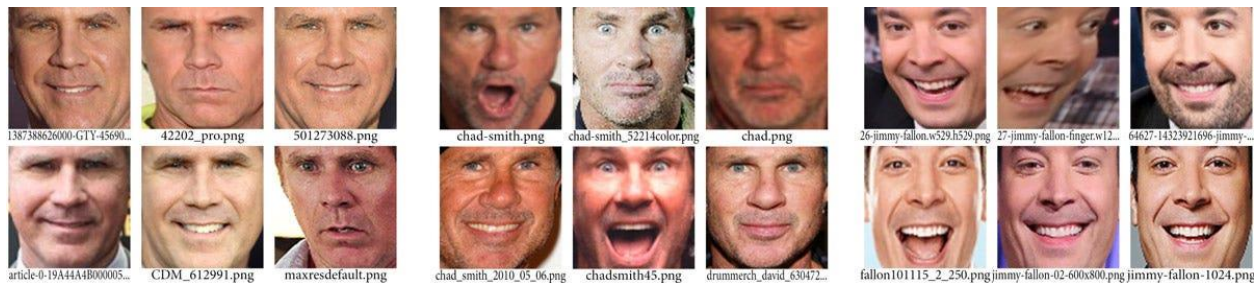
So run all face images through their pre-trained network to get the 128 measurements for each face. Here is the measurements for our test image:

Input Image		128 Measurements Generated from Image			
	0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014	0.020976085215807
	0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807	-0.00052163278451189
	0.030809439718723	-0.01981477253139	0.10801389068365	-0.1318951100111	-0.0059557510539889
	0.036050599068403	0.065554238855839	0.0731306001544	0.043374512344599	-0.053343612700701
	-0.097486883401871	0.1226262897253	-0.029626874253154	-0.078198105096817	-0.076289616525173
	-0.0066401711665094	0.036750309169292	-0.15958009660244	0.12369467318058	0.056418422609568
	-0.14131525158882	0.14114324748516	-0.031351584941149	0.089727647602558	-0.0085843298584223
	-0.048540540039539	-0.061901587992907	-0.15042643249035	-0.022388197481632	-0.020696049556136
	-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.050584398210049	-0.072376452386379
	-0.061418771743774	-0.074287034571171	-0.065365232527256	-0.045013956725597	-0.012774495407939
	0.046741496771574	0.0061761881224811	0.14746543765068	-0.013955107890069	-0.17898085713387
	-0.12113650143147	-0.21055991947651	0.0041091227903962	-0.072600327432156	0.0050511928275228
	0.061606746166945	0.11345765739679	0.021352224051952	-0.014829395338893	-0.043765489012003
	0.061989940702915	0.19372203946114	-0.086726233363152	-0.012062266469002	0.012774495407939
	0.10904195904732	0.084853030741215	0.09463594853878	0.069833360612392	0.11638788878918
	-0.019414527341723	0.0064811296761036	0.21180312335491	-0.015336792916059	0.10281457751989
	0.15245945751667	-0.16582328081131	-0.035577941685915	-0.082041338086128	
	-0.12216668576002	-0.0072777755558491	-0.036901291459799		
	0.083934605121613	-0.059730969369411	-0.070026844739914		
	0.087945111095905	0.11478432267904	-0.089621491730213		
	-0.021407851949334	0.14841195940971	0.078333757817745		
	-0.018298890441656	0.049525424838066	0.13227833807468		
	-0.011014151386917	-0.051016297191381	-0.14132921397686		
	0.0093679334968328	-0.062812767922878	-0.13407498598099		
	0.058139257133007	0.0048638740554452	-0.039491076022387		
	-0.024210374802351	-0.11443792283535	0.071997955441475		
	-0.057223934680223	0.014683869667351	0.05228154733777		
	0.023535015061498	-0.081752359867096	-0.031709920614958		
	-0.0098039731383324	0.037022035568953	0.11009479314089		
	0.020220354199409	0.12788131833076	0.18632389605045		
	0.0040337680839002	-0.094398014247417	-0.11768248677254		
	0.051597066223621	-0.10034311562777	-0.040977258235216		

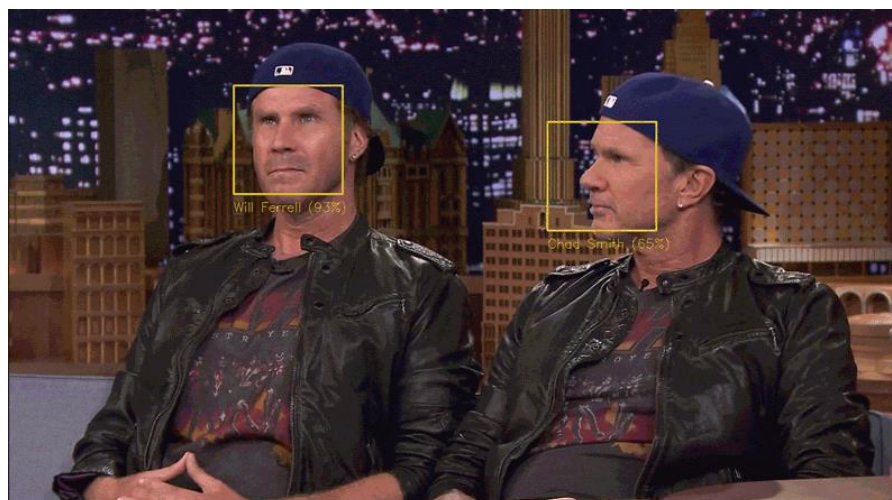
Step 4: Finding the person's name from the encoding:

The last step is the easiest step in the whole process. Now it is required to find the person in the database of known people who has the closest measurements to our test image. The approach used is simple linear SVM classifier. All needs to be done is train a classifier that can take in the measurements from a new test image and tell which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person!

First, train a classifier with the embeddings of about 20 pictures each of Will Ferrell, Chad Smith and Jimmy Fallon:



Then run the classifier on every frame (Snippet taken from YouTube video of Will Ferrell and Chad Smith pretending to be each other on the Jimmy Fallon show):



This approach can recognize the faces even when they are quite similar and is even able to detect the faces with a side angle.

CODE IMPLEMENTATION

4.1 Libraries:

```
1 import cv2
2 import numpy as np
3 import face_recognition
4 import os
5 from datetime import datetime
```

1. `import cv2`: This line imports the OpenCV library, which is a popular open-source computer vision library used for various image and video processing tasks.

2. `import numpy as np`: This line imports the NumPy library and assigns it the alias `np`. NumPy is a fundamental package for scientific computing with Python and provides support for handling multidimensional arrays and mathematical operations.

3. `import face_recognition`: This line imports the `face_recognition` library, which is a Python module for face recognition and facial feature detection. It provides easy-to-use functions for face detection, face landmarks, and face recognition tasks.

4. `import os`: This line imports the `os` module, which provides a way to interact with the operating system. It offers functions to manipulate file paths, directories, and other operating system-related tasks.

5. `from datetime import datetime`: This line imports the `datetime` class from the `datetime` module within the Python standard library. It allows you to work with dates and times, including getting the current date and time.

These import statements are essential for utilizing the functionalities provided by the respective libraries/modules in the code.

4.2 Data Retrieval:

```
7 path = 'ImagesAttendance'  
8 images = []  
9 classNames = []  
10 myList = os.listdir(path)
```

1. `path = 'ImagesAttendance'`: This line assigns the string `'ImagesAttendance'` to the variable `path`. It represents the path to the folder where the pre-stored images for attendance recognition are located.
2. `images = []`: This line initializes an empty list named `images`. This list will be used to store the images retrieved from the `'ImagesAttendance'` folder.
3. `classNames = []`: This line initializes an empty list named `classNames`. This list will be used to store the names of the individuals associated with the images. Each name will correspond to a specific image.
4. `myList = os.listdir(path)`: This line uses the `os.listdir()` function to retrieve a list of all files and directories present in the `'ImagesAttendance'` folder. The `os.listdir()` function returns a list containing the names of the entries in the specified path. This list of names is assigned to the variable `myList`.

3.3 Names for Attendance:

```
12 for cl in myList:
13     curImg = cv2.imread(f'{path}/{cl}')
14     images.append(curImg)
15     classNames.append(os.path.splitext(cl)[0])
16 print(classNames)
```

1. `for cl in myList:`: This line initiates a loop that iterates over each element in the `myList` list. Each element represents a file name or directory name in the `'ImagesAttendance'` folder.

2. `curImg = cv2.imread(f'{path}/{cl}')`: This line uses the `cv2.imread()` function from the OpenCV library to read an image file. The file path is constructed using f-string formatting, where `path` represents the `'ImagesAttendance'` folder path and `cl` represents the current file name in the loop. The resulting image is assigned to the variable `curImg`.

3. `images.append(curImg)`: This line appends the `curImg` image to the `images` list. This list will store all the images retrieved from the `'ImagesAttendance'` folder.

4. `classNames.append(os.path.splitext(cl)[0])`: This line uses the `os.path.splitext()` function to split the file name (`cl`) into the base name and extension. The `[0]` index is used to extract only the base name (without the extension). This base name represents the name of the individual associated with the image. The extracted name is then appended to the `classNames` list.

4. ``print(classNames)``: This line prints the ``classNames`` list, which contains the names of the individuals associated with the images from the ``ImagesAttendance`` folder.

4.4 Finding Encodings:

```
18 def findEncodings(images):  
19     encodeList = []  
20     for img in images:  
21         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
22         encode = face_recognition.face_encodings(img)[0]  
23         encodeList.append(encode)  
24     return encodeList
```

1. ``encodeList = []``: This line initializes an empty list named ``encodeList``. This list will store the face encodings of the images.

2. ``for img in images:``: This line initiates a loop that iterates over each image in the ``images`` list.

3. ``img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)``: This line converts the color space of the image from BGR (Blue, Green, Red) to RGB (Red, Green, Blue). The face recognition library ``face_recognition`` expects RGB images for processing.

4. ``encode = face_recognition.face_encodings(img)[0]``: This line uses the ``face_encodings`` function from the ``face_recognition`` library to compute the face encoding of the image. The function takes in the RGB image as input and returns a list of face encodings. In this case, it retrieves the first (and typically only) face encoding from the list and assigns it to the variable ``encode``.

5. ``encodeList.append(encode)``: This line appends the face encoding ``encode`` to the ``encodeList`` list.
6. ``return encodeList``: This line returns the ``encodeList`` containing the face encodings of all the images.

4.5 Mark Attendance:

```
26 def markAttendance(name):
27     with open('Attendance.csv','r+') as f:
28         myDataList = f.readlines()
29         nameList = []
30         for line in myDataList:
31             entry = line.split(',')
32             nameList.append(entry[0])
33         if name not in nameList:
34             now = datetime.now()
35             dtString = now.strftime('%d-%m-%Y')
36             dtString1 = now.strftime('%H:%M:%S')
37             f.writelines(f'\n{name},{dtString},{dtString1}')
```

1. ``encodeList = []``: This line initializes an empty list named ``encodeList``. This list will store the face encodings of the images.
2. ``for img in images:``: This line initiates a loop that iterates over each image in the ``images`` list.

3. `img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`: This line converts the color space of the image from BGR (Blue, Green, Red) to RGB (Red, Green, Blue). The face recognition library `face_recognition` expects RGB images for processing.

4. `encode = face_recognition.face_encodings(img)[0]`: This line uses the `face_encodings` function from the `face_recognition` library to compute the face encoding of the image. The function takes in the RGB image as input and returns a list of face encodings. In this case, it retrieves the first (and typically only) face encoding from the list and assigns it to the variable `encode`.

5. `encodeList.append(encode)`: This line appends the face encoding `encode` to the `encodeList` list.

6. `return encodeList`: This line returns the `encodeList` containing the face encodings of all the images.

4.6 WebCam Access:

```
39 encodeListKnown = findEncodings(images)
40 print('Encoding Complete')
41
42 cap = cv2.VideoCapture(0)
```

1. `encodeListKnown = findEncodings(images)`: This line calls the `findEncodings()` function to compute the face encodings of the `images` list. The resulting face encodings are stored in the `encodeListKnown` list.

2. ``print('Encoding Complete')``: This line prints the message "Encoding Complete" to indicate that the face encoding process has finished.

3. ``cap = cv2.VideoCapture(0)``: This line creates a ``VideoCapture`` object named ``cap`` that captures video from the default camera (index 0). This will be used for capturing live video frames for face recognition or any other video processing tasks.

4.7 Resize:

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

1. ``success, img = cap.read()``: This line captures a frame from the video source specified by the ``cap`` object. It returns two values: ``success``, which indicates whether the frame was successfully read, and ``img``, which represents the captured frame.

2. ``imgS = cv2.resize(img,(0,0),None,0.25,0.25)``: This line resizes the captured frame (``img``) to a smaller size using the ``cv2.resize()`` function. The parameters ``(0,0)`` indicate that the size of the output image is calculated automatically based on the scaling factors provided as the last two parameters (``0.25`` in this case). The scaling factors ``0.25, 0.25`` reduce the size of the image to one-fourth of its original dimensions.

3. ``imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)``: This line converts the color space of the resized image (``imgS``) from BGR to RGB using the ``cv2.cvtColor()`` function. The face recognition library ``face_recognition`` typically expects RGB images for processing.

4.8 Finding face locations:

```
49     facesCurFrame = face_recognition.face_locations(imgS)
50     encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFrame)
```

1. `facesCurFrame = face_recognition.face_locations(imgS)`: This line uses the `face_locations` function from the `face_recognition` library to detect the locations of faces in the resized image `imgS`. The function returns a list of tuples, where each tuple represents the coordinates (top, right, bottom, left) of a detected face in the image.

2. `encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFrame)`: This line uses the `face_encodings` function from the `face_recognition` library to compute the face encodings of the detected faces in the resized image `imgS`. The function takes in the image and the face locations as inputs and returns a list of face encodings.

4.9 Recognizing:

```
52     for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):
53         matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
54         faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
55         matchIndex = np.argmin(faceDis)
```

1. `for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):` This line initiates a loop that iterates over the elements of `encodesCurFrame` and `facesCurFrame` simultaneously using the `zip()` function. Each `encodeFace` represents a face encoding, and each `faceLoc` represents the corresponding face location.

2. `matches = face_recognition.compare_faces(encodeListKnown, encodeFace)`: This line uses the `compare_faces` function from the `face_recognition` library to compare the `encodeFace` (the face encoding of the current face) against the `encodeListKnown` (the list of known face encodings). The function returns a list of boolean values indicating whether a match is found between the current face and each known face encoding.

3. `faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)`: This line uses the `face_distance` function from the `face_recognition` library to calculate the Euclidean distance between the `encodeFace` (the face encoding of the current face) and each face encoding in `encodeListKnown`. The function returns an array of distance values.

4. `matchIndex = np.argmin(faceDis)`: This line uses the NumPy `argmin` function to find the index of the smallest (minimum) value in the `faceDis` array. It assigns this index to the variable `matchIndex`, representing the index of the closest match in the known face encodings list.

4.10 Matching:

```
57     if matches[matchIndex]:
58         name = classNames[matchIndex].upper()
59
60         y1,x2,y2,x1 = faceLoc
61         y1,x2,y2,x1 = y1*4,x2*4,y2*4,x1*4
62         cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
63         cv2.rectangle(img,(x1,y2-35),(x2,y2),(0,255,0),cv2.FILLED)
64         cv2.putText(img,name,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
65         markAttendance(name)
```

1. `if matches[matchIndex]:` This line checks if the `matchIndex` corresponds to a match in the `matches` list. If there is a match, the code proceeds to execute the following lines.

2. `name = classNames[matchIndex].upper()`: This line retrieves the name associated with the matching face from the `classNames` list and assigns it to the variable `name`. The name is converted to uppercase using the `upper()` method.

3. `y1,x2,y2,x1 = faceLoc`: This line unpacks the coordinates of the detected face location from the `faceLoc` tuple into separate variables.

4. `y1,x2,y2,x1 = y1*4,x2*4,y2*4,x1*4`: This line scales up the face location coordinates by a factor of 4. It is done to match the coordinates with the original size of the image, as the captured frame (`img`) was resized earlier.

5. `cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)`: This line draws a rectangle around the detected face on the `img` frame. It uses the `cv2.rectangle()` function from OpenCV, specifying the top-left and bottom-right coordinates of the rectangle, color (in BGR format), and line thickness.

6. `cv2.rectangle(img,(x1,y2-35),(x2,y2),(0,255,0),cv2.FILLED)`: This line draws a filled rectangle below the face rectangle to serve as a background for displaying the name. The rectangle is positioned based on the bottom-left and top-right coordinates, using a slightly reduced `y` value to provide some space between the name and the face rectangle.

7. `cv2.putText(img,name,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)`:

This line puts the `name` text on the image. It uses the `cv2.putText()` function to display the name on the image frame, specifying the text, position (top-left corner), font, font scale, color (in BGR format), and thickness.

8. `markAttendance(name)`: This line calls the `markAttendance()` function to mark the attendance for the identified person by passing the `name` as a parameter.

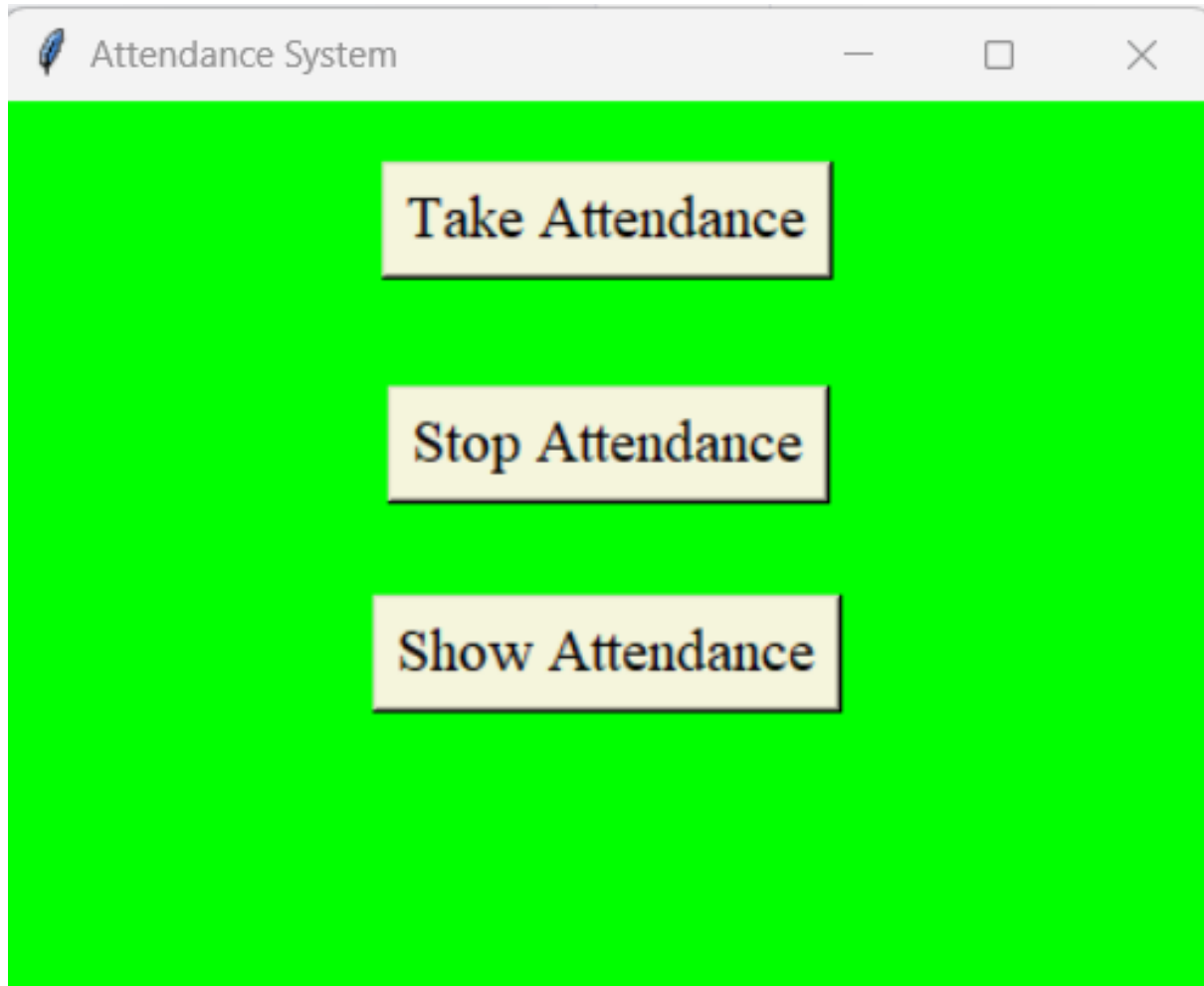
4.11 Webcam:

```
67     cv2.imshow('Webcam',img)
68     cv2.waitKey(1)
```

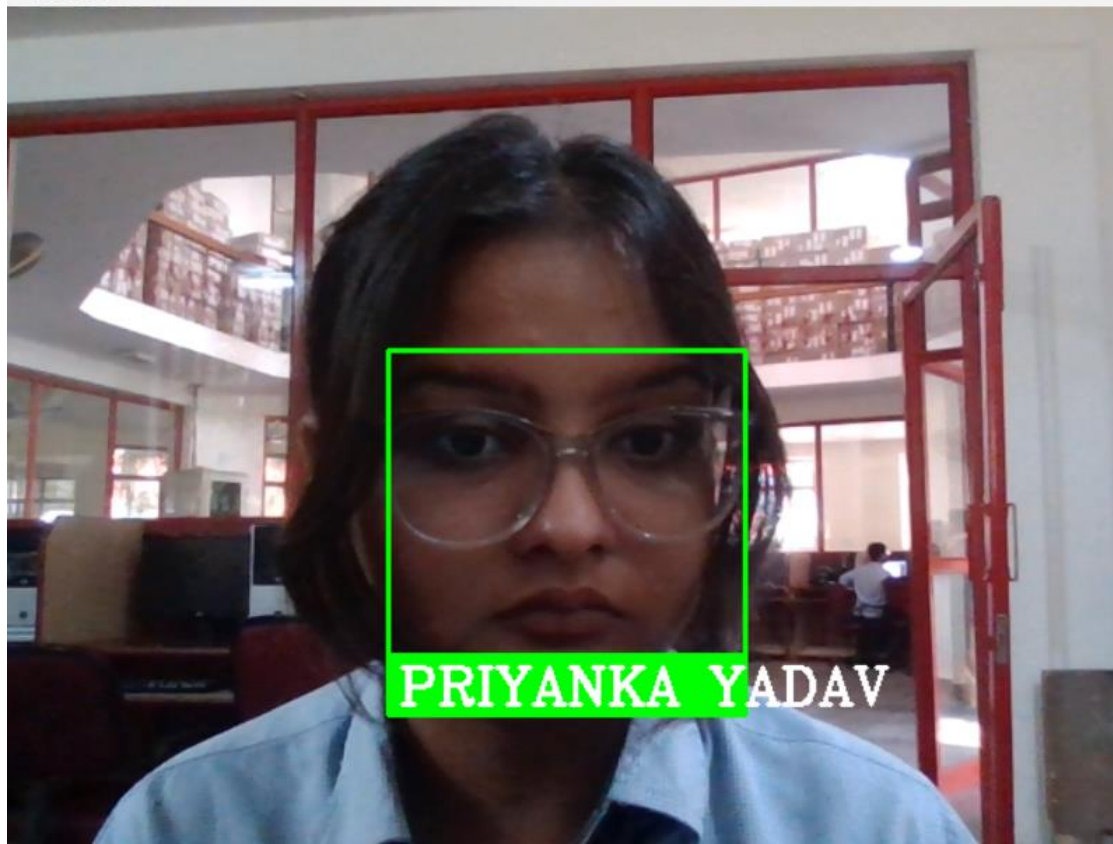
1. `cv2.imshow('Webcam', img)`: This line displays the `img` frame in a window with the title "Webcam". It uses the `cv2.imshow()` function from the OpenCV library to show the image.

2. `cv2.waitKey(1)`: This line waits for a keyboard event for 1 millisecond. It allows the window to stay open and display the image until a key is pressed. The function returns the ASCII value of the key pressed.

4.12 Results:



Webcam



	A	B	C	
1	Name	Date	Time	
2				
3	SHRISHTI J	07-07-2023	00:54:41	
4	PRIYANKA	07-07-2023	00:54:51	
5	Priyanka Y	07-07-2023	08:24:18	
6				

Conclusion

The Face Recognition-based Attendance System, developed using Python and OpenCV, offers a reliable and efficient solution for automating attendance management through face recognition technology. By leveraging computer vision algorithms and machine learning techniques, the system provides accurate and real-time attendance tracking, eliminating the need for manual processes and reducing administrative burden.

With the hardware requirements consisting of a camera and a standard computer system, and the software requirements including Python, OpenCV, and face recognition libraries, the Face Recognition-based Attendance System can be implemented on various operating systems.

In conclusion, the Face Recognition-based Attendance System offers a reliable, accurate, and user-friendly approach to attendance management. By harnessing the power of Python and OpenCV, this system streamlines the attendance tracking process, enhances efficiency, and ensures data security, contributing to a more streamlined and efficient educational or organizational environment.

References

- <https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78>
- <https://opencv.org/>
- <https://datagen.tech/guides/face-recognition/face-recognition-with-python/>
- <https://www.studocu.com/in>
- <https://www.wikipedia.org/>