Automatic Attendance Management System

A Project Report submitted in the partial fulfillment of the

Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

M. Teja	**(20471A05L8)**
N. Pavan kumar	**(21475A0515)**
N. Venkata Siva Suneel	**(21475A0503)**

Under the esteemed guidance of

Dr. S.V.N. Sreenivasu Rao M.Tech., Ph.D

Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE

(AUTONOMOUS)

(Affiliated to JNTUK, Kakinada, Approved by AICTE &Thrice Accredited by NBA)

2023- 2024

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "Automatic Attendance Management System" is a bonafide work done by the team M. Teja (20471A05L8), N. Pawan Kumar (21475A0515), N.Venkata Siva Suneel (21475A0515) **in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2023-2024.**

PROJECT GUIDE

Dr. S. V.N. Sreenivasu Rao, M.Tech., Ph.D.,
Professor

PROJECT CO-ORDINATOR

M. Sireesha, M.Tech.(Ph.D),
Assoc. Professor

HEAD OF THE DEPARTMENT

Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,

HOD

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We wish to express my thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman sri **M.V.Koteswara Rao,B.Sc.,** who took keen interest in us in every effort throughout this course. We owe out sincere gratitude to our beloved principal **Dr.M.Sreenivasa Kumar, M.Tech., Ph.D., MISTE., FIE(I).,** for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr.S.V.N Sreenivasu Rao, M.Tech.,Ph.D.,** HoD of CSE department and also to our guide **Dr.S.N.Tirumala Rao, M.Tech.,Ph.D.,** of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Ms Sireesha .M, M.Tech.,** Associate professor & Project coordinator of the project for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfully completing our project.

By

M. Teja (20471A05L8)
N. Pavan Kumar (21475A0515)
N.Venkata Siva Suneel (21475A0503)

ABSTRACT

Facial recognition technology plays a crucial role in various applications, from enhancing security at banks and organizations to streamlining attendance tracking in public gatherings and educational institutions. Traditional methods of attendance marking, such as signatures, names, and biometrics, can be time-consuming and error-prone. To address these challenges, a smart attendance system is proposed, leveraging Deep Learning, Convolutional Neural Networks (CNN), and the OpenCV library in Python for efficient face detection and recognition.

The system utilizes advanced algorithms, including Eigen faces and fisher faces [4], to recognize faces accurately. While deep learning models excel with large datasets, they may not perform optimally with few samples. By comparing input faces with images in the dataset, the system automatically updates recognized names and timestamps into a CSV file, which is then sent to the respective organization's head. Additionally, the system allows users to upload a single photo or a group photo, and it returns matched photos as output using a CNN. This feature enhances the system's flexibility and usability, providing users with a convenient way to identify and track individuals in various scenarios.

INDEX

S.NO.	CONTENT	PAGE NO
1.	Introduction	01
2.	Literature Survey	06
2.1	History	07
2.2	How deep learning works	09
2.3	Importance Of Deep Learning	11
2.4	Convolutional Neural Networks	12
2.5	Recurrent Neural Networks	13
2.6	Long Short-Term Memory Networks	13
2.7	Generative Adversarial Networks	13
2.8	How Deep Learning works in face recognition	13
3.	Face Recognition	15
3.1	Process of face recognition	15
3.2	How Does Facial Recognition Work	16
3.3	Face Recognition Algorithms	18
3.3.1	Eigen faces	18
4.	Convolutional Neural Network	20
4.1	Convolutional Layer	21
4.2	Libraries Used	24
4.2.1	OpenCV	24
4.2.2	NumPy	25
4.2.3	Tensor flow	25
4.2.4	Keras	25
4.2.5	Pygame	26
5.	System Requirements	26
5.1	Hardware Requirements	26
5.2	Software Requirements	26
6.	System Analysis	27
6.1	Scope of Project	27
6.2	Data Preprocessing	27
7.	Design Analysis	30
8.	Implementation	31
8.1	Collecting the Dataset	31

8.2 Face Detection Module 31
8.3 Classification Model 32
8.4 Training 33
8.5 Evaluation 34
8.6 Testing with Test Data 34
8.7 Implementation code 34
9. Result Analysis 58
10. Future Scope 62
11. Conclusion 62
12. References 63

LIST OF FIGURES

S.NO. LIST OF FIGURES PAGE NO

1. Fig 2 Deep Learning 06
2. Fig 2.1 History 07
3. Fig 2.2 Flow chart of deep learning 09
4. Fig 2.2.3 Machine Learning Vs Deep Learning 10
5. Fig 2.8 Facial Encoding 14
6. Fig 3.2 Face Recognition Flow 17
7. Fig 3.3 Sample Data set 20
8. Fig 4.5.1 Flatten Layer 22
9. Fig 4.5.2 Dense Layer 23
10. Fig 4.5.3 Dropout Layer 23
11. Fig 6.2.2 after Preprocessing 29.
12. Fig 9.1: web interface 58
13. Fig 9.2 Authentication 59
14. Fig 9.3 User Registration 59
15. Fig 9.4 Image's Capturing 60
16. Fig 9.5 Attendance Record in excel sheet 60
17. Fig9.6 Single Photo Matching 61
18. Fig 9.7 Group photo Matching 61

1. INTRODUCTION

Facial recognition technology has indeed revolutionized security and attendance tracking systems, offering a more efficient and accurate alternative to traditional methods such as signatures and biometrics. This technology finds extensive application in banks, organizations, public gatherings, and educational institutions, where maintaining security and tracking attendance are paramount.

Facial recognition technology works by analyzing and identifying unique facial features, such as the distance between the eyes, the shape of the nose, and the contours of the face. These features are then converted into a mathematical representation known as a faceprint, which is stored in a database. When a person's face is captured by a camera, the system compares the captured faceprint with the stored faceprints to determine if there is a match.

One of the key advantages of facial recognition technology is its accuracy. Unlike traditional methods such as signatures and biometrics, which can be forged or manipulated, facial recognition technology is difficult to deceive. This makes it an ideal solution for security and attendance tracking systems, where accuracy and reliability are critical.

Facial recognition technology is also incredibly fast. In a matter of seconds, a facial recognition system can capture, analyze, and identify a person's face, making it ideal for high-traffic areas such as airports, train stations, and sports stadiums. This speed and efficiency make it easier for organizations to manage large crowds and ensure the safety of their premises.

Furthermore, facial recognition technology is highly scalable. It can be easily integrated into existing security systems, making it a cost-effective solution for organizations looking to enhance their security measures. Additionally, facial recognition technology can be used in conjunction with other security measures, such as access control systems and surveillance cameras, to provide a comprehensive security solution.

In addition to security applications, facial recognition technology is also being used in attendance tracking systems. In educational institutions, for example, facial recognition technology can be used to track student attendance, eliminating the need for manual attendance taking. This not only saves time but also reduces the risk of errors.

Facial recognition technology is also being used in banking and financial institutions to enhance security. By using facial recognition technology, banks can verify the identity of customers, making it more difficult for fraudsters to gain access to accounts. This can help prevent identity theft and other forms of fraud, protecting both customers and banks.

Facial recognition technology has revolutionized security and attendance tracking systems, offering a more efficient and accurate alternative to traditional methods. Its accuracy, speed, scalability, and versatility make it an ideal solution for a wide range of applications, from security and access control to attendance tracking and customer verification. As technology continues to advance, facial recognition technology is likely to play an increasingly important role in ensuring the safety and security of individuals and organizations alike.

To further enhance the capabilities of existing systems, a cutting-edge smart attendance system is proposed. This system leverages the power of Deep Learning, Convolutional Neural Networks (CNNs), and the OpenCV library in Python, incorporating advanced algorithms like Eigenfaces and Fisherfaces for precise face detection and recognition, thereby ensuring accurate attendance marking. This sophisticated approach to attendance management and security surveillance represents a significant leap forward, combining the strengths of several advanced technologies to achieve unparalleled accuracy and efficiency.

Deep Learning, a subset of machine learning, mimics the workings of the human brain in processing data and creating patterns for use in decision making. It is particularly effective for tasks that involve recognizing patterns, such as image and speech recognition. Convolutional Neural Networks (CNNs), a class of deep neural networks, are specifically designed to process pixel data and are used extensively in image recognition tasks. These networks apply a series of filters to the input images to learn hierarchical features, which makes them incredibly effective for facial recognition tasks.

By utilizing CNNs, the proposed smart attendance system can accurately identify unique facial features among individuals, even in varying lighting conditions and angles. This level of precision ensures that the system is highly reliable, significantly reducing the likelihood of false positives or negatives that could compromise attendance accuracy.

The OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a common infrastructure for computer vision applications and accelerates the use of machine perception in commercial products. By integrating OpenCV with Python, the system gains access to numerous algorithms and tools for image processing, which is crucial for real-time face detection and recognition.

Eigenfaces and Fisherfaces are two such algorithms that have been instrumental in advancing facial recognition technology. Eigenfaces refers to an approach that uses principal component analysis (PCA) to reduce the dimensionality of the facial images, focusing on the most significant features that distinguish one face from another. Fisherfaces extends this approach by using Linear Discriminant Analysis (LDA) to maximize the ratio of between-class scatter to the within-class scatter in the data, enhancing the system's ability to distinguish between faces even further.

The proposed smart attendance system integrates these technologies to create a robust solution capable of accurately identifying individuals and marking their attendance in real-time. This system can be deployed across various sectors, including educational institutions, corporate offices, and public gatherings, to streamline attendance processes and enhance security measures.

In educational settings, such a system can automate the attendance process, freeing up valuable time for teachers and ensuring accurate attendance records. In the corporate world, it can enhance security by controlling access to sensitive areas and providing accurate logs of employee attendance, which can be invaluable for HR management and security oversight. At public gatherings, it can improve security and efficiency by quickly identifying attendees and potentially alerting authorities to the presence of individuals who may pose a security risk.

The integration of Deep Learning, CNNs, OpenCV, and sophisticated algorithms like Eigenfaces and Fisherfaces into smart attendance systems brings a multitude of benefits, chief among them being high accuracy, efficiency, scalability, and versatility. These systems are designed to precisely identify individuals, drastically reducing the margin for error and enhancing security measures in various settings. Their real-time processing capabilities ensure quick handling of large data volumes, making them suitable for environments with high attendance throughput. Furthermore, the flexibility of these systems allows for easy scaling, accommodating organizations

of varying sizes and needs across sectors such as education, corporate, and public events. Despite these advantages, the implementation of such advanced biometric systems necessitates careful consideration of privacy concerns and the development of stringent regulations to govern the ethical use of biometric data. Securing the data collected is crucial to prevent potential misuse or unauthorized access, ensuring the technology serves to enhance safety and convenience without compromising individual privacy.

The proposed smart attendance system represents a significant advancement in the field of facial recognition and attendance management. By combining Deep Learning, CNNs, the OpenCV library, and advanced algorithms like Eigenfaces and Fisherfaces, this system offers a highly accurate, efficient, and scalable solution that can be adapted to a wide range of applications. As technology continues to evolve, such systems are set to become an integral part of our approach to security and attendance management, marking a new era of efficiency and effectiveness in these domains.

The inherent challenge of deep learning models, including those based on Convolutional Neural Networks (CNNs), is their performance with limited data. Typically, these models require extensive datasets to train effectively, capturing the wide variance within human faces. However, the proposed smart attendance system addresses this limitation ingenitiously, ensuring robust performance even when faced with a few samples. This adaptability is crucial for environments where capturing extensive datasets for each individual might be impractical or invasive.

At the core of the system's design is an automatic update mechanism that compares input faces with images in the dataset. Upon recognition, the system automatically updates recognized names and timestamps into a Comma-Separated Values (CSV) file. This file acts as a dynamic attendance log, which can be easily accessed and analyzed for various purposes, such as tracking attendance over time, identifying patterns in behavior, or ensuring security compliance. This real-time processing and updating mechanism not only enhance the system's efficiency but also ensure its data remains current, reflecting the latest interactions without manual intervention.

A standout feature of the proposed system is its ability to handle both single and group photo uploads. Users can upload a photo of an individual or a photo containing multiple people. The system, powered by a CNN, then processes this image, identifying and matching faces against the stored dataset. This process involves detecting faces within a group photo, isolating them, and then running the recognition algorithm on each detected face individually.

The ability to handle group photos significantly enhances the system's flexibility and usability. It allows for quick and convenient processing in various scenarios, such as class photos in

educational settings, team photos in corporate environments, or photos from gatherings and events. This feature not only saves time by processing multiple individuals simultaneously but also offers a user-friendly approach to attendance tracking and security monitoring, eliminating the need for individual check-ins.

The system's output further adds to its usability. After processing the uploaded photos, it returns matched photos along with the identification information of the individuals recognized. This immediate feedback can be invaluable in various scenarios, from verifying attendance in educational and corporate settings to enhancing security by quickly identifying individuals in a crowd.

Moreover, the user-friendly interface ensures that users, regardless of their technical expertise, can interact with the system effectively. This accessibility broadens the potential user base, making it suitable for a wide range of applications beyond those initially envisioned.

The integration of an automatic updating mechanism and the ability to process both single and group photos significantly enhances the proposed smart attendance system's flexibility, efficiency, and usability. By addressing the challenge of limited data samples with real-time data processing and providing convenient features for users, the system stands out as a sophisticated solution in the realm of facial recognition and attendance tracking. As technology continues to evolve, such innovations will pave the way for more accessible, accurate, and user-friendly systems, transforming how we approach identity verification, security, and data management in various sectors.

2. LITERATURE SURVEY

2.1 DEEP LEARNING

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It's on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture. A formal definition of deep learning is- neurons. Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbours. The question here is how we recreate these neurons in a computer. So, we create an artificial structure called an artificial neural net where we have nodes or neurons. We have some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

Face recognition is the problem of identifying and verifying people in a photograph by their face. It is a task that is trivially performed by humans, even under varying light and when faces are changed by age or obstructed with accessories and facial hair. Nevertheless, it is remained a challenging computer vision problem for decades until recently. Deep learning methods are able to leverage very large datasets of faces and learn rich and compact representations of faces, allowing modern models to first perform as-well and later to outperform the face recognition capabilities of humans.

Fig 2.1-Deep Learning

1. **Deep Neural Network** (DNN) – It is a neural network with a certain level of complexity (having multiple hidden layers in between input and output layers). They are capable of modeling and processing non-linear relationships.
2. **Deep Belief Network (DBN)** – It is a class of Deep Neural Network. It is multi-layer belief networks.

   Steps:

   a. Learn a layer of features from visible units using Contrastive Divergence algorithm.

   b. Treat activations of previously trained features as visible units and then learn features of features.

   c. Finally, the whole DBN is trained when the learning for the final hidden layer is achieved.

3. **Recurrent** (perform same task for every element of a sequence) **Neural Network** – Allows for parallel and sequential computation. Similar to the human brain (large feedback network of connected neurons). They are able to remember important things about the input they received and hence enables them to be more precise.

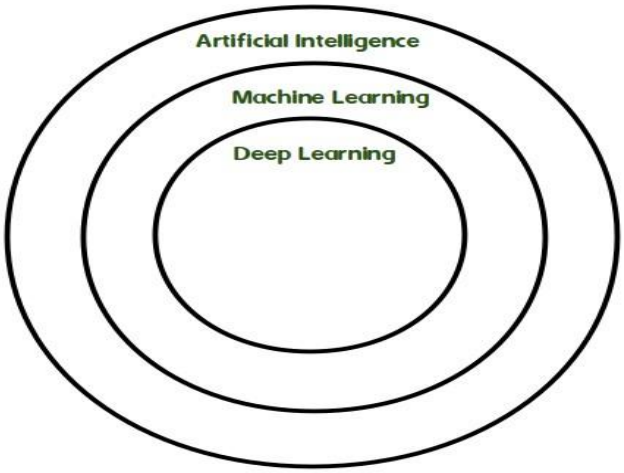

Fig 2.2-Architecture of Deep Learning

2.1 HISTORY:

Fig

2.1 History of Deep Learning

Deep Learning, as a branch of Machine Learning, employs algorithms to process data and imitate the thinking process, or to develop abstractions. Deep Learning (DL) uses layers of algorithms to process data, understand human speech, and visually recognize objects. Information is passed through each layer, with the output of the previous layer providing input for the next layer. The first layer in a network is called the input layer, while the last is called an output layer. All the layers between the two are referred to as hidden layers. Each layer is typically a simple, uniform algorithm containing one kind of activation function.

Feature extraction is another aspect of Deep Learning. Feature extraction uses an algorithm to automatically construct meaningful "features" of the data for purposes of training, learning, and understanding. Normally the Data Scientist, or programmer, is responsible for feature extraction.

The history of Deep Learning can be traced back to 1943, when Walter Pitts and Warren McCulloch created a computer model based on the neural networks of the human brain. They used a combination of algorithms and mathematics they called "threshold logic" to mimic the thought process. Since that time, Deep Learning has evolved steadily, with only two significant breaks in its development. Both were tied to the infamous Artificial Intelligence winters.

The earliest efforts in developing Deep Learning algorithms came from Alexey Grigoryevich Ivakhnenko

(developed the Group Method of Data Handling) and Valentin Grigor′evich Lapa (author of Cybernetics and Forecasting Techniques) in 1965. They used models with polynomial (complicated equations) activation functions, that were then analyzed statistically. From each layer, the best statistically chosen features were then forwarded on to the next layer (a slow, manual process).

The first "convolutional neural networks" were used by Kunihiko Fukushima. Fukushima designed neural networks with multiple pooling and convolutional layers. In 1979, he developed an artificial neural network, called Neocognitron, which used a hierarchical, multilayered design. This design allowed the computer the "learn" to recognize visual patterns. The networks resembled modern versions, but were trained with a reinforcement strategy of recurring activation in multiple layers, which gained strength over time. Additionally, Fukushima's design allowed important features to be adjusted manually by increasing the "weight" of certain connections.

2.2 How deep learning works:

First, we need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). Second, we need to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Third, Choose the Deep Learning Algorithm appropriately. Fourth, Algorithm should be used while training the dataset. Fifth, Final testing should be done on the dataset.

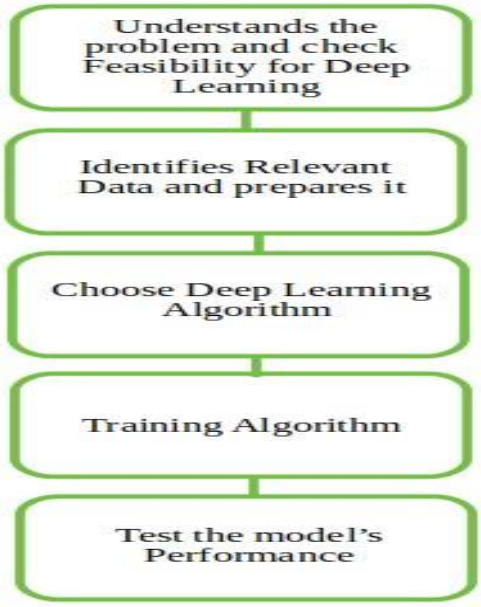

Fig 2.2- Flow chart of Deep Learning

Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network. Any Deep neural network will consist of three types of layers: • Input Layer

- Hidden Layer
- Output Layer

1.The Input layer:

It receives all the inputs and the last layer is the output layer which provides the desired output.

2. Hidden Layers:

All the layers in between these layers are called hidden layers. There can be n number of hidden layers. The hidden layers and perceptions in each layer will depend on the use-case you are trying to solve.

3. Output Layers:

It provides the desired output.

Machine Learning vs Deep Learning:

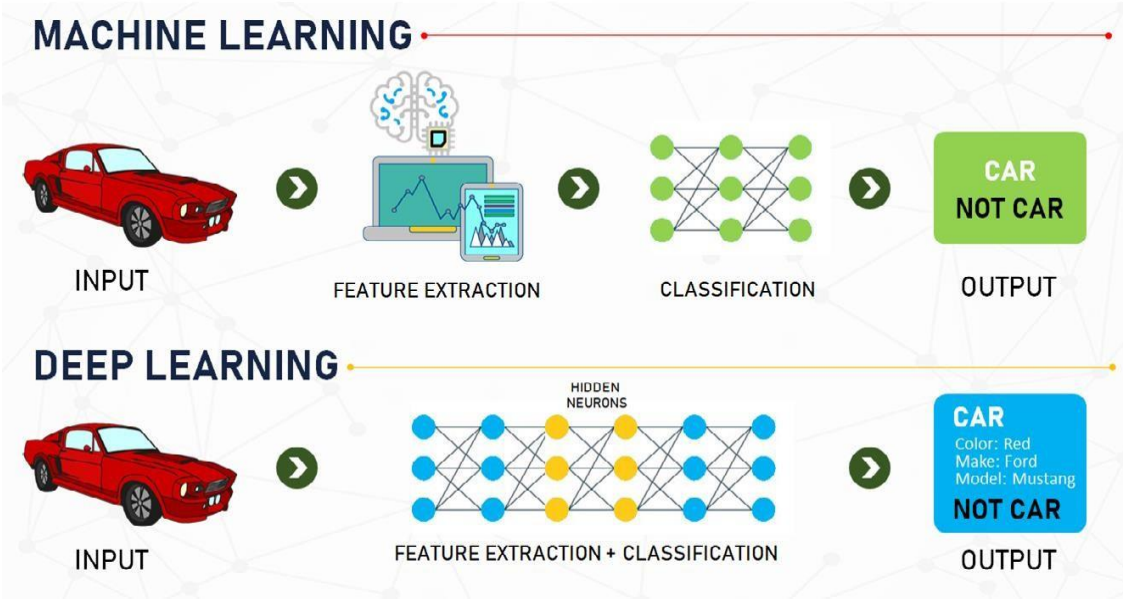

Fig 2.2.3- Machine Learning vs Deep Learning

Machine Learning:

Machine learning is a subset, an application of Artificial Intelligence (AI) that offers the ability to the system to learn and improve from experience without being programmed to that level. Machine Learning uses data to train and find accurate results. Machine learning focuses on the development of a computer program that accesses the data and uses it to learn from themselves.

Deep Learning:

Deep Learning is a subset of Machine Learning where the artificial neural network, the recurrent neural network comes in relation. The algorithms are created exactly just like machine learning but it consists of many more levels of algorithms. All these networks of the algorithm are together called as the artificial neural network. In much simpler terms, it replicates just like the human brain as all the neural networks are connected in the brain, exactly is the concept of deep learning. It solves all the complex problems with the help of algorithms and its process.

2.3 IMPORTANCE OF DEEP LEARNING:

Deep learning holds significant importance across various domains due to its ability to learn complex patterns and representations directly from data. Here are several reasons why deep learning is considered important:

Fig 2.3.1 Applications of Deep Learning

1. High Performance: Deep learning models, particularly deep neural networks, have demonstrated superior performance in various tasks such as image classification, object detection, speech recognition, natural language processing, and more. They often outperform traditional machine learning algorithms when dealing with large and complex datasets.

2. Feature Learning: Deep learning architectures can automatically learn hierarchical representations of data, extracting relevant features at multiple levels of abstraction. This eliminates the need for manual feature engineering, allowing the model to discover intricate patterns and relationships in the data on its own.

3. Scalability: Deep learning models can scale effectively to handle large volumes of data. With advancements in hardware accelerators (such as GPUs and TPUs) and distributed

computing frameworks, training deep neural networks on massive datasets has become feasible, enabling the development of more sophisticated models.

4. Versatility: Deep learning techniques can be applied to a wide range of tasks across different domains, including computer vision, natural language processing, speech recognition, healthcare, finance, autonomous vehicles, and more. This versatility makes deep learning a valuable tool for solving diverse real-world problems.

5. Continual Advancements: Deep learning research is continuously evolving, leading to the development of novel architectures, optimization algorithms, regularization techniques, and training methodologies. This ongoing progress drives innovation and allows deep learning models to achieve state-of-the-art performance in various applications.

6. Unstructured Data Handling: Deep learning excels at processing unstructured data types such as images, audio, video, and text, which are prevalent in today's digital world. By effectively learning from raw data, deep learning models can uncover insights and patterns that may be difficult to extract using traditional methods.

7. Decision Making: Deep learning models can assist in decision-making processes by providing accurate predictions, classifications, or recommendations based on learned patterns from data. This has implications across sectors such as healthcare (diagnosis and treatment planning), finance (fraud detection and risk assessment), and marketing (customer segmentation and personalized recommendations).

Overall, the importance of deep learning lies in its ability to leverage large amounts of data to learn complex patterns, enabling the development of powerful and versatile AI systems with applications across various industries and domains.

2.4 CONVOLUTIONAL NEURAL NETWORKS (CNNS):

CNNs are a subset of deep learning models created especially to handle input that resembles a grid, like photographs. They are made up of several layers, such as fully connected, pooling, and convolutional layers. CNNs are frequently employed for problems involving object detection, picture segmentation, and image classification because they are excellent at removing spatial hierarchies of features from input data.

2.5 RECURRENT NEURAL NETWORKS (RNNS):

RNNs are an additional category of deep learning models that are frequently employed for processing sequential data, including time series data and natural language sequences. They are useful for tasks like sentiment analysis, language translation, and speech recognition because of their feedback connections, which enable information to remain over time.

2.6 LONG SHORT-TERM MEMORY (LSTM) NETWORKS:

A specific kind of RNN called an LSTM network deal with the vanishing gradient issue that arises when RNNs are trained on lengthy data sequences. With its more complex design of memory cells and gates controlling the input flow, long-range dependencies may be captured by LSTMs, which also allow them to learn from sequences with different time delays. For tasks like speech recognition and language modeling that require sequential data with long-term dependencies, they work especially well.

2.7 GENERATIVE ADVERSARIAL NETWORKS (GANS):

Deep learning models called GANs are made up of two neural networks—the discriminator and the generator—that are trained in competition with one another concurrently.

While the discriminator assesses the veracity of both actual and synthetic samples, the generator creates synthetic data samples. Image generation, image-to-image translation, and data augmentation are among the tasks that GANs are being employed.

2.8 How Deep Learning works in face recognition

Deep learning is one of the most novel ways to improve face recognition technology. The idea is to extract face embeddings from images with faces. Such facial embeddings will be unique for different faces. And training of a deep neural network is the most optimal way to perform this task.

Depending on a task and timeframes, there are two common methods to use deep learning for face recognition systems:

Use pre-trained models such as **dlib**, **DeepFace**, **FaceNet**, and others. This method takes less time and effort because pre-trained models already have a set of algorithms for face

recognition purposes. We also can fine-tune pre-trained models to avoid bias and let the face recognition system work properly.

Develop a neural network from scratch. This method is suitable for complex face recognition systems having multi-purpose functionality. It takes more time and effort, and requires millions of images in the training dataset, unlike a pre-trained model which requires only thousands of images in case of transfer learning.

Encoding the faces using OpenCV and deep learning

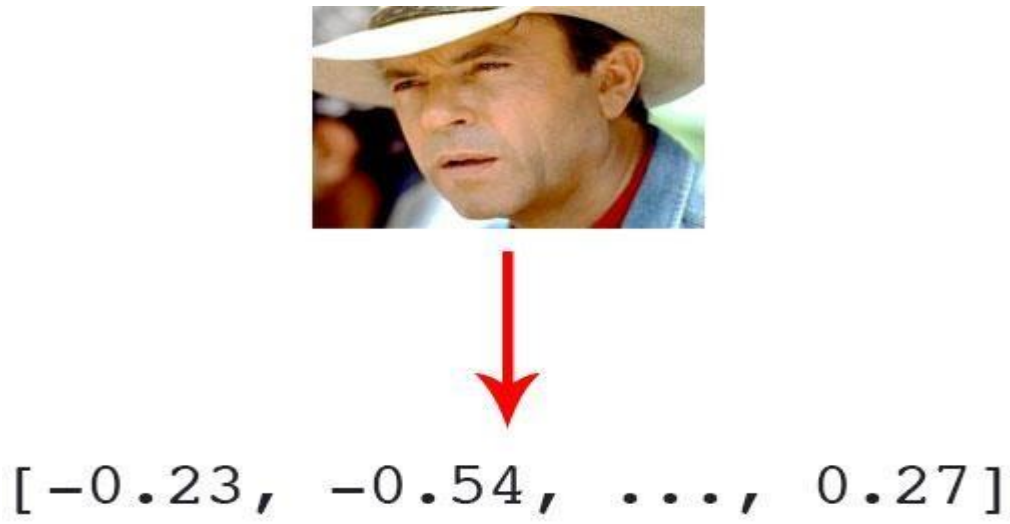

Fig 2.8- Facial Encodings

Facial recognition via deep learning and Python using the face_recognition module method generates a 128-d real-valued number feature vector per face.

Before we can recognize faces in images and videos, we first need to quantify the faces in our training set. Keep in mind that we are not actually training a network here — *the network has* **already been trained** *to create 128-d embeddings* on a dataset of ~3 million images.

We certainly **could** train a network from scratch or even fine-tune the weights of an existing model but that is more than likely overkill for many projects. Furthermore, you would need a **lot** of images to train the network from scratch.

Instead, it's easier to use the pre-trained network and then use it to construct 128-d embeddings for each of the 218 faces in our dataset.

Then, during classification, we can use a simple k-NN model + votes to make the final face classification.
Other traditional machine learning models can be used here as well.

CHAPTER 3: Face Recognition

Face detection using deep learning in OpenCV involves leveraging pre-trained deep learning models to detect faces in images or video streams. One of the commonly used pre-trained models in OpenCV is based on the Single Shot Multibox Detector (SSD) framework with a MobileNet base. This model is efficient and provides accurate face detection results.

The process begins by loading the pre-trained model using the cv2.dnn.readNetFromCaffe function, which takes the path to the model's prototxt file and its caffemodel file. The prototxt file contains the network architecture, while the caffemodel file contains the learned weights of the network.

Once the model is loaded, an image is preprocessed to meet the input requirements of the model. This preprocessing involves resizing the image to a fixed size (e.g., 300x300 pixels) and applying mean subtraction and scaling to normalize the pixel values.

The preprocessed image is then passed through the network using the net.setInput function, and the net.forward function is called to perform forward pass inference. This process generates a set of detections, each representing a detected face in the image.

These detections are then post-processed to filter out weak detections based on a confidence threshold. The remaining detections are used to draw bounding boxes around the detected faces on the original image.

Overall, face detection using deep learning in OpenCV provides a robust and efficient way to detect faces in images, making it suitable for a wide range of applications, including photography, biometric security, and human-computer interaction.

3.1 Process of face recognition

Face recognition is often described as a process that first involves four steps; they are: **face detection**, face alignment, feature extraction, and finally face recognition.

1. **Face Detection**. Locate one or more faces in the image and mark with a bounding box.
2. **Face Alignment**. Normalize the face to be consistent with the database, such as geometry and Photometrics.
3. **Feature Extraction**. Extract features from the face that can be used for the recognition task.
4. **Face Recognition**. Perform matching of the face against one or more known faces in a prepared database.

A given system may have a separate module or program for each step, which was traditionally the case, or may combine some or all of the steps into a single process.

The facial recognition process can be split into two major stages: processing which occurs before detection involving face detection and alignment and later recognition is done using feature extraction and matching steps.

1. **FACE DETECTION:**

The primary function of this step is to conclude whether the human faces emerge in a given image, and what is the location of these faces. The expected outputs of this step are patches which contain each face in the input image. In order to get a more robust and easily designable face recognition system. Face alignment is performed to rationalise the scales and orientation of these patches.

2. **FEATURE EXTRACTION:**

Following the face detection step the extraction of human face patches from images is done. After this step, the conversion of face patch is done into vector with fixed coordinates or a set of landmark points.

3. **FACE RECOGNITION:**

The last step after the representation of faces is to identify them. For automatic recognition we need to build a face database. Various images are taken foe each person and their features are extracted and stored in the database. Then when an input image is fed the face detection and feature extraction is performed and its feature to each face class is compared and stored in the database.

3.2 How Does Facial Recognition Work?

The computer algorithm of facial recognition software is a bit like human visual recognition. But if people store visual data in a brain and automatically recall visual data once needed, computers should request data from a database and match them to identify a human face.

In a nutshell, a computerized system equipped by a camera, detects and identifies a human face, extracts facial features like the distance between eyes, a length of a nose, a shape of a forehead and cheekbones.

Then, the system recognizes the face and matches it to images stored in a database.

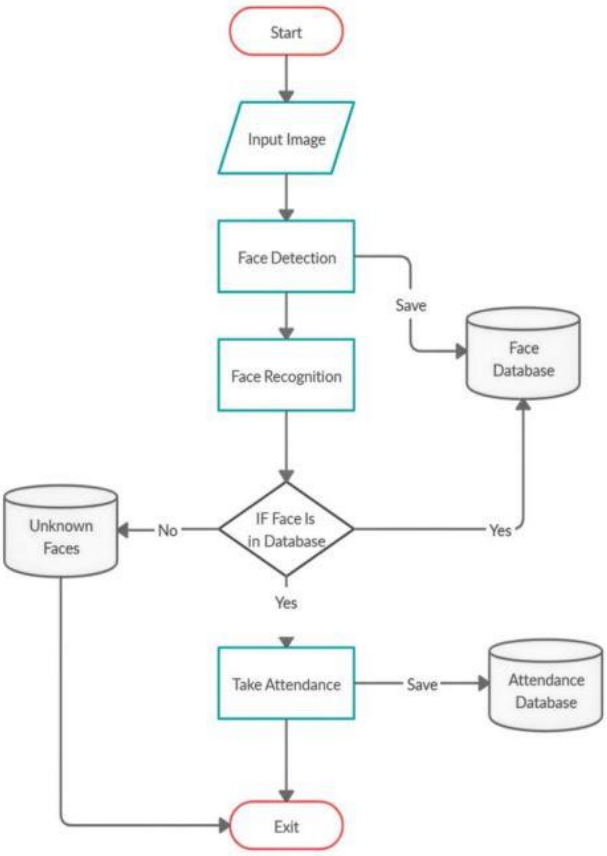

Fig 3.2- Face Recognition Process Flow

Three face recognition tasks:

- **Face Matching**: Find the best match for a given face.
- **Face Similarity**: Find faces that are most similar to a given face.
- **Face Transformation**: Generate new faces that are similar to a given face.

Matching requires that the candidate matching face image be in some set of face images selected by the system. Similarity detection requires in addition to matching that images of faces be found which are similar to a recalled face this requires that the similarity measure used by the recognition system closely match the similarity measures used by humans Transformation applications require that new images created by the system be similar to human recollections of a face.

Two main modes for face recognition, are as:

1.Face Verification.

A one-to-one mapping of a given face against a known identity (e.g. *is this the person?*).

2.Face Identification.

A one-to-many mapping for a given face against a database of known faces (e.g. *who is this person?*). There is a difference between Face verification and Face recognition some people often take them as same. It is important to know this difference.

3.3 Face Recognition Algorithms

The face recognition algorithm is used to find the characteristics which accurately represents a picture with the features which are already extracted, scaled, and converted into grey scale. There are various algorithms used for facial recognition. Some of them are as follows:

3.3.1.Eigen faces:

Face recognition systems are built on the idea that each person has a particular face structure, and using the facial symmetry, computerized face-matching is possible. The work on face recognition has begun in the 1960's, the results of which are being used for security in various institutions and firms throughout the world. The images must be processed correctly for computer-based face recognition. The face and its structural properties should be identified carefully, and the resulting image must be converted to two-dimensional digital data. An efficient algorithm and a database which consists of face images are needed to solve the face recognition problem. In the recognition process, an eigenface is formed for the given face image, and the Euclidian distances between this eigenface and the previously stored eigenfaces are calculated. The eigenface with the smallest Euclidian distance is the one the person resembles the most. Simulation results are shown. Simulations have been done using the Matlab program. The success rate for the large database used is found to be 94.74 percent. The face recognition system is similar to other biometric systems. The idea behind the face recognition system is the fact that each individual has a unique face. Similar to the fingerprint, the face of an individual has many structures and features unique to that individual. An automatic face recognition system is based on facial symmetry. Face authentication and face identification are challenging problems. The fact that in the recent past, there have been more and more commercial, military and institutional applications, makes the face recognition systems a popular subject. To be reliable, such systems have to work with high precision and accuracy. In a face recognition system, the database consists of the images of the individuals that the system has to recognize. If possible, several images of the same individual should be included in the database. If the images are selected so that they account for varying facial expressions, lighting conditions, etc., the solution of the problem can be found more easily as compared to the case where only a single image of each individual is stored in the database. A face recognition algorithm processes the captured image and compares it to the images stored

in the database. If a match is found, then the individual is identified. If no match is found, then the individual is reported as unidentified. The challenges of face recognition are:

1.Shifting and scaling of the image

2.Differences in the facial look (different angle, pose, hairstyle, makeup, mustache, beard, etc.), x Lighting,

Aging.

The algorithm has to work successfully even with the above challenges.

The basis of the eigenfaces method is the Principal Component Analysis (PCA). Eigenfaces and PCA have been used by Sirovich and Kirby to represent the face images efficiently [11]. They have started with a group of original face images, and calculated the best vector system for image compression. Then Turk and Pentland applied the Eigenfaces to face recognition problem [12]. The Principal Component Analysis is a method of projection to a subspace and is widely used in pattern recognition. An objective of PCA is the replacement of correlated vectors of large dimensions with the uncorrelated vectors of smaller dimensions. Another objective is to calculate a basis for the data set. Main advantages of the PCA are its low sensitivity to noise, the reduction of the requirements of the memory and the capacity, and the increase in the efficiency due to the operation in a space of smaller dimensions. The strategy of the Eigenfaces method consists of extracting the characteristic features on the face and representing the face in question as a linear combination of the so called ‘eigenfaces’ obtained from the feature extraction process. The principal components of the faces in the training set are calculated. Recognition is achieved using the projection of the face into the space formed by the eigenfaces. A comparison on the basis of the Euclidian distance of the eigenvectors of the eigenfaces and the eigenface of the image under question is made. If this distance is small enough, the person is identified. On the other hand, if the distance is too large, the image is regarded as one that belongs to an individual for which the system has to be trained.

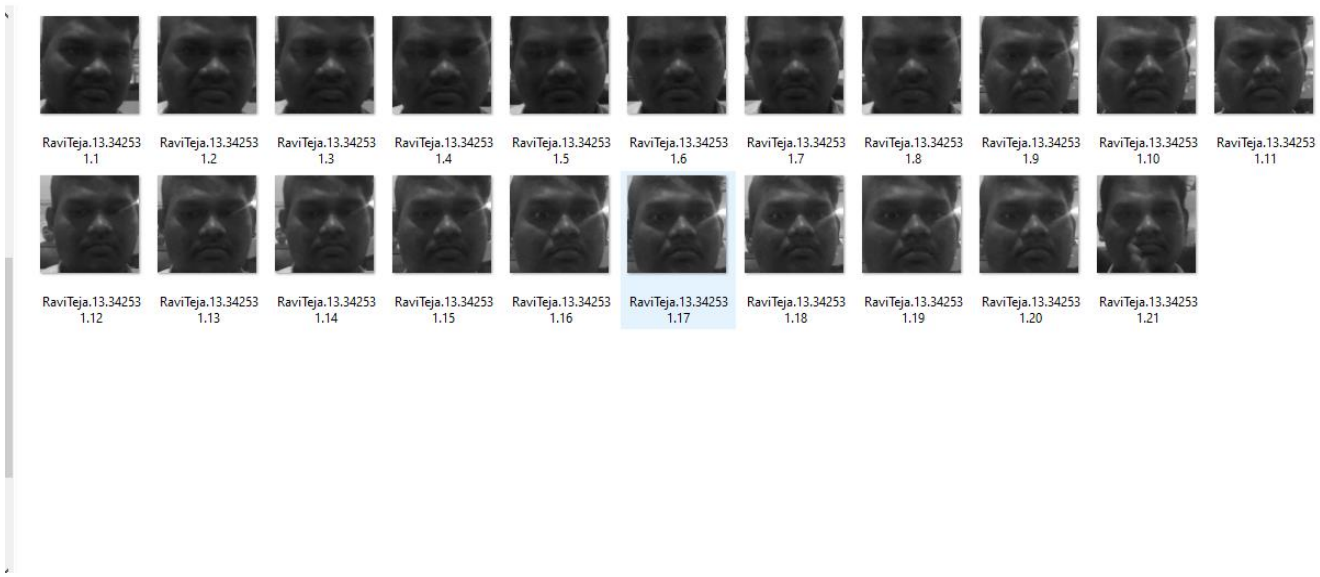

Fig 3.3 Sample data set

4 Convolutional Neural Network:

A Convolutional Neural Network (InceptionV3/CNN) is a Deep Learning model that can recognize different objects and aspects in an input image and distinguish between them by assigning weights and biases that can be learned. When considering alternative classification methods, the pre-processing needed for an InceptionV3 is significantly less. While filters are manually designed in more archaic approaches, InceptionV3s can learn these traits and filters with sufficient training. Inspired by the structure of the visual cortex, the architecture of an InceptionV3 is comparable to the pattern of connections between neurons in the human brain. Only in a small area of the visual field known as the Receptive Field do individual neurons react to inputs. To fill the whole visual field, a group of these fields overlap. By using the appropriate filters, an InceptionV3 may effectively capture the spatial and temporal dependencies present in an image. Because fewer parameters are used in the design and weights can be reused, it fits the picture dataset more accurately.

Stated differently, the network can be trained to comprehend the image's complexity more effectively. In CNN, the phrase "Convolution" refers to the mathematical function, a unique type of linear operation in which two functions are multiplied to create a third function that expresses how the other function changes the form of the first function. To put it simply, an output that is utilized to extract features from the image is produced by multiplying two matrices-representable images. CNN architecture is divided into two primary sections:

1. A convolution tool that, through a process known as feature extraction, divides and recognizes the image's numerous features for analysis.

2. A fully connected layer that makes use of the convolution process' output to forecast the image's class using the features that were extracted in earlier phases.

4.1 Convolutional Layer:

The main components of convolutional neural networks are convolutional layers. The different features from the input photos are first extracted using this layer. This layer performs the convolutional mathematical process between an input image and a filter with a specified MxM size. The dot product between the filter and the portions of the input image with regard to the filter's size (MxM) is calculated by swiping the filter over the image. If the filter is intended to identify a particular kind of feature in the input, it can find that feature wherever in the image by applying the filter consistently throughout the whole input image. The output, known as the feature map, provides us with details about the image, including its edges and corners. This feature map is later supplied to additional layers so that they can identify more features from the input image.

There are four types of layers that make up the CNN which are the convolutional layers:

1. Flatten Layer
2. Dense Layer (activation function relu)
3. Dropout Layer
4. Dense Layer (activation function softmax)

I. Flatten Layer:

The flatten layer is essential to getting the extracted ocular features ready for classification in the driver drowsiness detection project. The flatten layer acts as a link between the convolutional and pooling layers—which identify and emphasize pertinent characteristics from the input images—and the fully connected layers that come after.

The output of the convolutional and pooling layers is specifically reshaped into a onedimensional array or vector by the flatten layer. The convolutional layers' spatial data is combined in this transition to create a format that can be fed into the neural network's dense, fully connected layers.

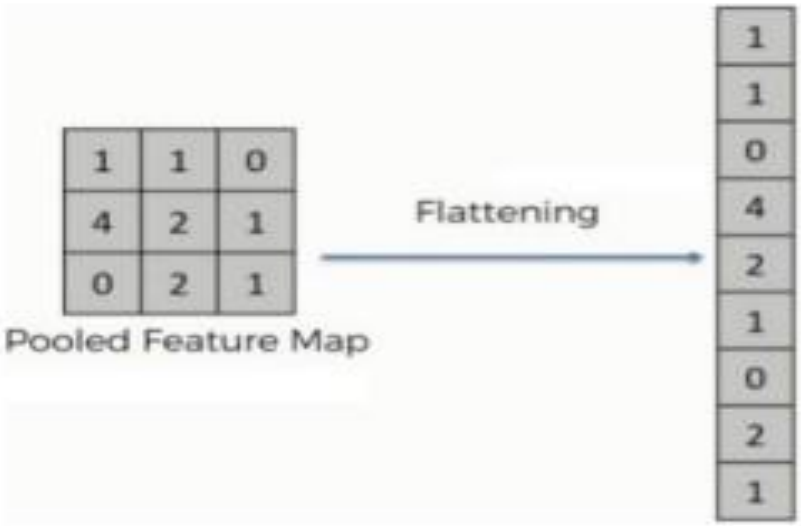

Fig 4.5.1 **Flatten Layer**

The flatten layer makes ensuring that all extracted features are efficiently used for classification by flattening the multidimensional feature maps produced by the convolutional layers. By learning intricate patterns and correlations within the feature space, the ensuing dense layers are able to accurately predict the eye status (open or closed) based on the input images.

II. Dense Layer (relu):

Rectified Linear Unit, or ReLU, is a dense layer activation function that is essential to adding non-linearity to the neural network architecture in the driver drowsiness detection project. The dense layers, which come after the flatten layer, are in charge of discovering intricate correlations and patterns in the feature space that are taken from the input images. Applying the ReLU activation function inside these dense layers causes non-linearity to be introduced by setting any negative input values to zero while maintaining positive values.

Furthermore, the scalability and efficacy of the neural network model are enhanced by the ReLU activation function's computational efficiency and simplicity. Through ReLU's mitigation of the vanishing gradient issue, which is frequently experienced with other activation functions such as sigmoid or tanh, deep neural network training becomes more stable and effective.

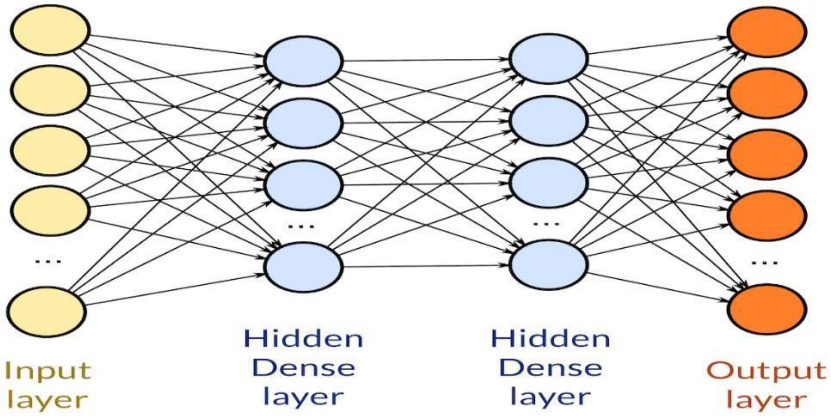

Fig 4.5.2 **Dense Layer**

The model can efficiently learn and categorize patterns indicative of tirednekss from the input picture data thanks to the ReLU activation function within the dense layers in the context of driver drowsiness detection. ReLU improves the overall accuracy and resilience of the sleepiness detection system by introducing non-linearity and capturing complex interactions. This ultimately improves road safety by quickly identifying indicators of driver weariness or inattention.

III. Dropout Layer:

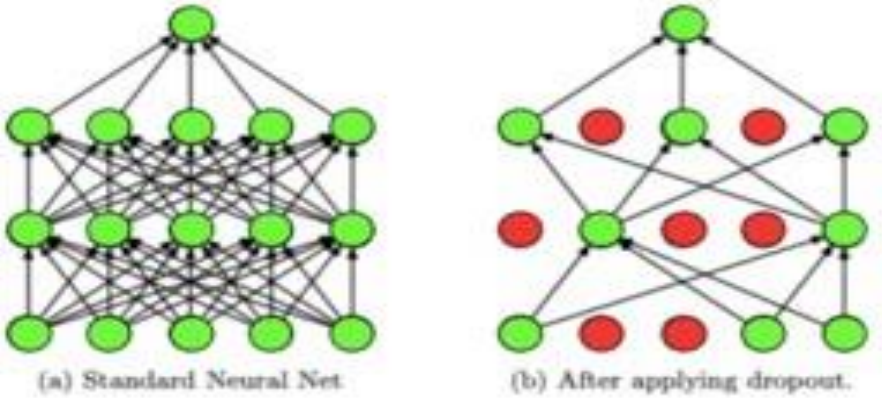

Fig 4.5.3 **Dropout Layer**

Overfitting in the training dataset may result from connecting every feature to the FC layer. Overfitting is the phenomenon where a certain model performs poorly when applied to new data since it performed so well on the training set. In order to solve this issue, a dropout layer is used, in which a small number of neurons are removed from the neural network during training,

reducing the size of the model. Thirty percent of the nodes in the neural network are randomly removed after a dropout of 0.3.

IV. Dense Layer (softmax):

The selection of the softmax activation function in the last dense layer of the driver drowsiness detection project is crucial for multi-class classification, which allows for the distinction of different eye states, such as open and closed. The dense layer with softmax activation, the last layer in the neural network architecture, converts the output of the layers before it into a probability distribution across a number of classes.

The network generates a vector of probabilities by applying softmax activation, where each element denotes the likelihood of the associated class. This allows the model to give a thorough evaluation of the likelihood of various eye states in the context of driver sleepiness detection. When the model identifies closed eyes, for example, it gives the "closed" class a high probability and gives lower probabilities to other classes, such "open" or "partially closed."

The softmax function is appropriate for multi-class classification jobs since it guarantees that the probabilities add up to one. Since the predicted class is associated with the highest probability, this attribute makes it possible to interpret the model's predictions intuitively. Softmax activation also makes it easier to compute the categorical cross-entropy loss, which is used as the training goal function and directs the optimization process toward precise classification.

4.2 LIBRARIES USED:

4.2.1 OpenCV (Open-Source Computer Vision Library):

An open-source software library for computer vision and machine learning is called OpenCV. In order to facilitate the use of machine perception in commercial goods and to give computer vision applications a common foundation, OpenCV was developed. OpenCV's BSD license facilitates easy code modification and usage for commercial enterprises. It supports Windows, Linux, Android, and Mac OS and includes interfaces in C++, Python, Java, and MATLAB. With a preference for real-time vision applications, OpenCV uses MMX and SSE instructions when they are available. CUDA and OpenCL interfaces with full functionality are now under active development. More than 500 algorithms exist, and around ten times as many functions either support or comprise those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

4.2.2 NumPy:

Numerical Python is referred to as NumPy. In the year 2005, Travis Oliphant founded NumPy. The core Python library for scientific computing is called NumPy. This Python library offers a multidimensional array object, a number of derived objects (such matrices and masked arrays), and a variety of quick operations procedures.

On arrays, encompassing a wide range of topics such as discrete Fourier transforms, fundamental linear algebra, basic statistical operations, random simulation, sorting, choosing, I/O, and much more. The nd array object is the central component of the NumPy package. It has numerous auxiliary functions that greatly simplify the use of the nd array. In data research, when resources and performance are critical, arrays are employed extensively. Because NumPy arrays are kept in memory continuously, as opposed to lists, processes can access and work with them with great efficiency. In computer science, this behavior is referred to as locality of reference. The primary factor that makes NumPy quicker than lists is this. It is also designed to function with the newest CPU architectures. A large portion of NumPy, a Python library, is written in C or C++, although the majority of the bits that need to compute quickly are written in Python.

4.6.3 Tensor flow:

An open-source software library is called TensorFlow. TensorFlow was initially created for machine learning and deep neural network research by scientists and engineers on the Google Brain Team within Google's Machine Intelligence research organization. However, the system is sufficiently general to be useful in a wide range of other domains as well.

Most programmers can access TensorFlow. Google's Tensorflow engine approaches problem-solving in a novel way. This special approach makes machine learning problems solvable issues with great efficiency. is a framework for defining and performing tensor-related computations, as the name suggests. Vectors and matrices are expanded to possibly greater dimensions as tensors. Tensors are internally represented as n-dimensional arrays of base datatypes by TensorFlow. The data type of every element in the Tensor is constant and has the same format. It is possible that only a portion of the shape—that is, the number of dimensions and size of each dimension—is known. If the forms of the inputs are also fully known, the majority of operations yield tensors of fully known shapes; but, in certain situations, it's only possible to find the shape of a tensor at graph execution time.

4.6.4 Keras:

An open-source library for deep learning (for neural networks) with a Python foundation is called Keras. 2015 saw the release of Keras. It can operate on top of Theano, Microsoft CNTK,

or TensorFlow. It is ideal for quick experimentation and is really easy to use and comprehend. It is intended to be quick and simple for the user to operate. Keras models can be executed on a CPU or a GPU. The greatest platform available for working with neural network models is Keras. Keras offers an API that is easy to use and understand even for beginners. One of Keras's advantages is that it can select any library for backend support. Several pretrained models are available in Keras, assisting the user in further enhancing the models the user is designing.

4.6.5 Pygame:

The cross-platform pygame module is primarily used for game development, while it can also be used to play audio files. There are no requirements for this; just install it using pip and execute it. On platforms, implementation is different. It makes use of App Kit, wind 11 winm on Windows. G Streamer on Linux and NS Sound on Mac OS X. sound.play() is a function in the pygame module. It is compatible with WAV and MP3 files.

5. SYSTEM REQUIREMENTS

5.1 HARDWARE REQUIREMENTS:

- System Type : Intel Core i3 or above
- Cache memory : 4MB(Megabyte)
- RAM : 8 gigabyte (GB)
- Bus Speed : 5 GT/s DBI2
- Number of cores : 2
- Number of threads : 4

5.2 SOFTWARE REQUIREMENTS:

- Operating System : Windows 10 Home, 64 bit Operating System
- Coding Language : Python
- Python distribution: Anaconda

6. SYSTEM ANALYSIS

6.1 SCOPE OF PROJECT:

The proposed model has the capability of detecting and recognizing different faces and images from the camera. The face recognized matching purpose we using another module OpenCV. It's a powerful library for computer vision tasks, and it should work well for your facial recognition system. The data set which contains the images are pre-trained and tested using deep learning so that the input images would be well detected. This method is secure enough, reliable and available for use. Further CNN adds robustness to the model and using this approach of training data, 71.34% recognition rate has been achieved. Deep learning has advantage over machine learning for other face recognition techniques. The resultant of this entire process is nothing but creating an attendance marking system in which the unique id, name and some more details of the recognized faces could be entered automatically into a CSV file. In further updating this attendance system can be taken towards web development by creating a website and marking the attendance of the people automatically into the website of the organization so that there would be no need to update or mark attendance manually. Additionally, the model's ability to find the best match photos from the dataset when uploading single or group photos enhances its utility and effectiveness. Presently this project is developed using Flask python and is successfully running on web. In the future upation, a mobile application will be developed in which each and every student are given access with unique login details so that they can track their status of attendance from anywhere round the globe.

6.2 DATA PREPROCESSING:

When getting picture datasets ready for deep learning applications, such as driver sleepiness detection, pre-processing is an essential first step. Its significance is in improving the data's quality, consistency, and applicability for deep learning models' later analysis. Preprocessing first ensures uniformity throughout the collection by addressing variances and anomalies in image quality and format. To improve model performance, this may entail scaling photos to a standard resolution, changing brightness and contrast levels, and eliminating noise or artifacts. Pre-processing the dataset to standardize it helps models learn from the input photos more efficiently, which improves accuracy and generalization.

Pre-processing also lowers biases and confounding variables in the dataset, which enhances the machine learning model's resilience and equity. For example, class imbalance problems that are frequently seen in real-world datasets can be addressed with strategies like data balancing, which guarantees that the model learns to effectively identify examples from all classes, including minority classes like drowsy states. In order to speed up learning and avoid overfitting, pre-processing may also entail deleting superfluous or unnecessary features using feature selection or dimensionality reduction techniques.

Here we used two pre-processing steps:

1. Blurriness Reduction
2. Noise Removal

I. Blurriness Reduction:

A popular method for smoothing pictures is called Gaussian blur, which successfully reduces noise and blur in the picture. The image seems smoother by attenuating highfrequency components that cause blur through the use of a Gaussian kernel to convolve the image.
This process is used when a preset threshold is used to determine whether the input image is blurry. The variance of the Laplacian operator applied to the image's grayscale version establishes the threshold. The image is subjected to Gaussian blur reduction if the estimated Laplacian variance is less than the given threshold, signifying considerable blur.

When it comes to driver drowsiness detection, crisp, clear photos help the computer identify and evaluate facial traits that indicate alertness or drowsiness more successfully. By promptly alerting drivers who may be at danger of fatigue-induced impairment, this enhances road safety by improving the sleepiness detection system's overall accuracy and dependability.
Therefore, in order to maximize the performance of machine learning models in image-based applications and guarantee resilience and dependability in real-world scenarios, blur removal pre-processing techniques like Gaussian smoothing must be included.

II. Noise Removal:

In image analysis jobs, noise reduction is an important pre-processing step because it improves the clarity and quality of visual data, which is necessary for further analysis and interpretation. The system can concentrate on identifying significant information pertinent to the current task, such detecting driver drowsiness, by eliminating noise from photos. Clear and noise-free photos in this situation allow for a more precise identification of the facial expressions and traits linked to tiredness, which enhances the overall efficacy and dependability of the drowsiness detection system.
Gaussian smoothing and other noise reduction pre-processing methods are incorporated into the code to make sure the input photos are suited for machine learning algorithms to analyze later.

This eventually improves performance and reliability in applications like driver safety monitoring by reducing the impact of noise and artifacts found in real-world image data. As a result, noise reduction is essential for improving the usefulness and quality of picture data, enabling more reliable and accurate analysis in a range of image-based applications.

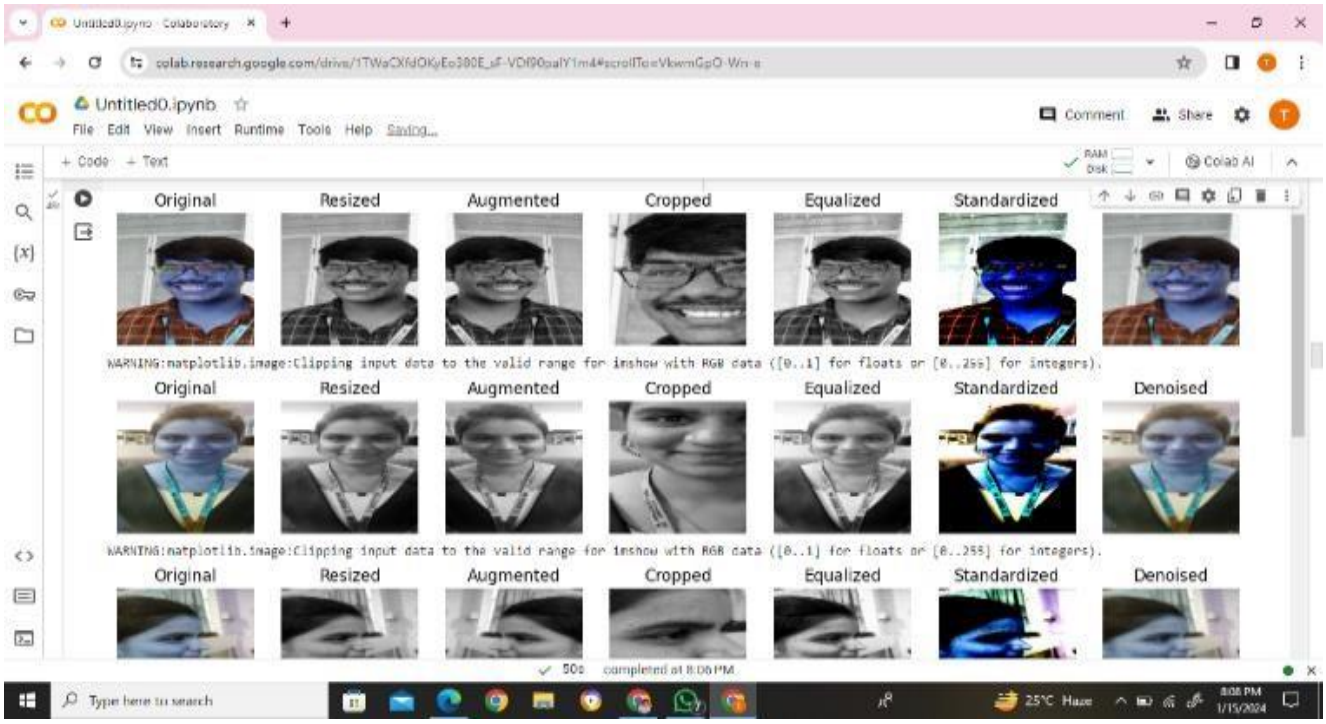

Fig 6.2.2: After Preprocessing

7. DESIGN ANALYSIS

The initial phase of the process involves extracting frames from a dataset, typically comprised of videos depicting drivers in various states, including instances of drowsiness and alertness. Subsequently, these extracted frames undergo preprocessing to enhance their quality for subsequent analysis. Preprocessing tasks may include resizing, converting to grayscale, and normalizing pixel values to ensure uniformity and clarity across the dataset.

Following preprocessing, the system engages in face detection, wherein it identifies and localizes faces within the frames. This step is crucial as it serves as the foundation for subsequent analysis. Once faces are detected, they are passed through a Convolutional Neural Network (CNN) classifier model. CNNs are well-suited for image recognition tasks and have likely been trained on a dataset containing labeled images of both drowsy and non-drowsy drivers.

Based on the features extracted from the facial images, the CNN classifier model provides a classification output indicating whether the driver is deemed "drowsy" or "not drowsy." Upon classification, if the model identifies the driver as drowsy, an alert is triggered. This alert can take various forms, such as visual or auditory cues, effectively notifying the driver of their current drowsy state, prompting them to take corrective action.

Fig 7.1 Design of System Process

8. IMPLEMENTATION

8.1 Collecting the Dataset:

Gathering the dataset is the initial stage. We require a vast amount of data to work on deep learning projects because model can't be done without it. Gathering and arranging the dataset is one of the most important steps in developing a deep learning project. A collection of photos with closed and open eyes can be found on the MRL official website under the name mrl_eye_dataset. This dataset includes numerous photos of people taken while operating a vehicle. There are additional pictures in this dataset of individuals sporting eyeglasses. Additionally, a dataset with a range of lighting conditions is included to ensure optimal model performance over time. We combined these images into 2 sets of images i.e., Closed eyes and Open eyes.

8.2 Face Detection Module:

Every image in the dataset has been preprocessed with this module. Two directories contain the images for the alert and sleepy classes. To list the directory names and every picture file inside each directory, we utilize the "os" module. After that, we have access to every image file in every directory. To read and resize the photos, we utilize OpenCV.

Since the background and other parts of the image are superfluous, only the face region is removed and provided to the classification model, rather than the full image. We do this by using a computer vision technique called face detection, which finds faces in digital photos.

The face detection algorithm known as the Haar Cascades Algorithm, or ViolaJones Algorithm, was put out by Paul Viola and Michael Jones in a 2001 article. How the algorithm operates: To extract objects, this algorithm makes advantage of Haar features.

After each iteration, the algorithm gradually modifies the window size by applying the features to the image's panes. A given window is not processed further if it is not identified as a face. It uses the cascade of classifiers notion rather than applying all the characteristics.

The features are applied one at a time, divided into various classifier steps. A window is only applied to the subsequent stage if it successfully completes the previous one;

otherwise, it is discarded and is no longer regarded as a face region. A face region is one that has made it through every phase. The beginning coordinates, width, and height of the detected face are returned by the open CV detect multiscale () function, and these are used to extract the face region from the corresponding image and resized to a fixed size 100x100.

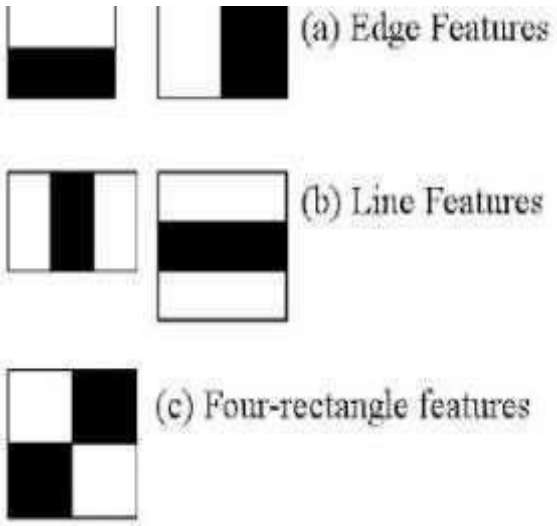

Fig 8.2.1 **Edge Features**

All of the resized images are added to one list, and the labels for each image are added to another list. The two lists are converted to numpy arrays, which we then provide to the classification model. The parameters image, scale factor, and min neighbours are utilized in the detect multiscale () function. The amount by which the image size is shrunk at each image scale is indicated by the scale Factor parameter. The number of neighbours that each candidate rectangle must have is specified by the min Neighbour option. The quality of faces that are detected is impacted by this attribute.

8.3 Classification Model:

In this module, we use the InceptionV3 architecture to build a Deep Learning Binary Classification model. Based on input visual data, binary classification tasks include predicting between two class labels, in this case, "alert" or "drowsy" states. A potent convolutional neural network (CNN) built for image classification applications is the InceptionV3 architecture. CNNs do particularly well on tasks requiring visual data because they are adept at capturing complex spatial relationships within images.

We use the pre-trained InceptionV3 model from the ImageNet dataset to create our model, and we set include top=False to remove the fully linked layers. The input form is

specified as (80,80,3), denoting images with three RGB color channels and a size of 80x80 pixels. The dimensions of the photographs in our collection are in line with this input shape.

Next, we add layers to the model one after the other to establish its architecture. To prepare it for input into the ensuing dense layers, the Flatten layer converts the multidimensional output from the convolutional layers into a one-dimensional array. Ten thousand neurons make up this layer, which is equivalent to the 10x10 feature map grid that was acquired from the preceding convolutional layers.

We add a Dense layer with 64 neurons and ReLU activation function after the Flatten layer. The convolutional layers' flattened feature vectors can be used by this dense layer to teach the model intricate patterns and representations. In order to avoid overfitting, we include a Dropout layer with a 0.5 dropout rate, which randomly removes half of the neurons during training in order to encourage generalization and avoid neuronal coadaptation.

The output layer with two neurons is then added, representing the two class labels ("alert" and "drowsy") in our binary classification problem. This layer uses the softmax activation function, which computes the probabilities for every class label and makes sure that the total of the predicted probabilities equals one.

Additionally, we set all layers in the basic model to non-trainable (trainable=False) in order to take advantage of the pre-trained InceptionV3 model's feature extraction capabilities without overfitting on our particular dataset. As a result, only the weights of the recently added layers can be updated during training, keeping the weights of the previously trained layers frozen. By using this tactic, the model's capacity to generalize to new data is enhanced and it is kept from memorizing the training set.

8.4 Training:

All of the available dataset is divided into train (80%) and test (100%) data after we shuffle the data. We run 5 epochs during training. A portion of the training data is used as validation data while the model is being trained. 20% of the data are used for verification. To train the model, the stochastic gradient descent (SGD) algorithm is applied. The validation data is not used to train the model; rather, it is used to assess the model's performance at the conclusion of each epoch. In our model, checkpoints are used to record the model's state every time a training improvement is noted. The model's weights are stored in the checkpoint. Following each improvement, the weights will be changed, and the best model will be saved when the training is completed.

8.5 Evaluation:

Measuring the model's performance after training is necessary. A classification model can be assessed using a variety of measures, including recall, accuracy, precision, and loss. To visualize the graphs of the various model metrics, we utilize Matplotlib.

8.6 Testing with test data:

Testing involves feeding the model previously unseen data in order to assess the model's capacity for generalization. The data was first divided into test and training sets. The model receives 20% of the data that it does not already know, and it uses this information to forecast the classes for the test data. Using the test data, we determine the model's accuracy and loss to assess its performance. Additionally, we determine the model's Precision and Recall values. Next, we carry out the same procedure using test data, which comprises 100% of our data.

Estimating the pictures that will be taken by the camera. We can use the model to predict the class of photos that are taken from the camera once it has been trained using the provided dataset. To get the pictures from the camera, we utilize OpenCV. We take picture frames from the camera all the time. Every frame that is collected undergoes the same preprocessing processes that are done to the dataset, namely: identifying the face in the image frame, obtaining the Region of Interest, and subsequently resizing the Region of Interest to a predetermined size (100x100).

Next, we prepare the photos for the model's input by converting them into an array format. The trained CNN Classification model can then be used to predict the labels for a series of photos. A voice alert is produced if the driver exhibits signs of sleepiness. The play sound module in Python is used to play an audio file.

8.7 IMPLEMENTATION CODE:

```
import os

import cv2

import matplotlib.pyplot as plt

def preprocess_dataset(dataset_dir, output_dir, target_size=(100, 100)):

    # Create the output directory if it doesn't exist
```

```
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    # Loop through each subdirectory (each person) in the dataset directory
    for person_name in os.listdir(dataset_dir):
        person_dir = os.path.join(dataset_dir, person_name)
        if os.path.isdir(person_dir):
            output_person_dir = os.path.join(output_dir, person_name)
            if not os.path.exists(output_person_dir):
                os.makedirs(output_person_dir)
            # Loop through each image file in the person's directory
            for filename in os.listdir(person_dir):
                image_path = os.path.join(person_dir, filename)
                output_image_path = os.path.join(output_person_dir, filename)
                # Read the image
                image = cv2.imread(image_path)
                # Preprocess the image (resize, normalize)
                if image is not None:
                    image = cv2.resize(image, target_size)  # Resize to target size
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
                    # Add additional preprocessing steps as needed (e.g., normalization)
                    # Save the preprocessed image
                    cv2.imwrite(output_image_path, image)
```

```
            else:
                print(f"Unable to read image: {image_path}")


def display_images(dataset_dir, num_images=10):
    fig, axes = plt.subplots(2, 5, figsize=(15, 6))
    axes = axes.flatten()
    images_count = 0

    # Loop through each subdirectory (each person) in the dataset directory
    for person_name in os.listdir(dataset_dir):
        person_dir = os.path.join(dataset_dir, person_name)
        if os.path.isdir(person_dir):
            # Loop through each image file in the person's directory
            for filename in os.listdir(person_dir):
                if images_count >= num_images:
                    break
                image_path = os.path.join(person_dir, filename)
                # Read the image
                image = cv2.imread(image_path)
                # Display the image
                if image is not None:
                    axes[images_count].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
                    axes[images_count].set_title(person_name)
```

```
                axes[images_count].axis('off')
                images_count += 1

    plt.tight_layout()
    plt.show()

# Define input and output directories for preprocessing
dataset_dir = "Images"  # Replace with the path to your dataset directory
output_dir = "preprocessed_dataset"  # Replace with the desired output directory
# Preprocess the dataset
preprocess_dataset(dataset_dir, output_dir)
print("Preprocessing successfully done.")
# Display 10 images from the preprocessed dataset
display_images(output_dir, num_images=10)
import os
import numpy as np
import cv2
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
```

```
from tensorflow.keras.callbacks import ModelCheckpoint


# Load preprocessed dataset

def load_dataset(dataset_dir):

    images = []

    labels = []

    for label, person_name in enumerate(os.listdir(dataset_dir)):

        person_dir = os.path.join(dataset_dir, person_name)

        for filename in os.listdir(person_dir):

            image_path = os.path.join(person_dir, filename)

            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            images.append(image)

            labels.append(label)

    return np.array(images), np.array(labels)


# Define dataset directory and output directory for the model

dataset_dir = "preprocessed_dataset"  # Replace with the path to your preprocessed dataset

output_dir = "trained_model"  # Replace with the desired output directory for the trained model


# Load preprocessed dataset

images, labels = load_dataset(dataset_dir)
```

```
# Split dataset into training and validation sets

X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)



# Normalize pixel values to [0, 1]

X_train = X_train / 255.0

X_val = X_val / 255.0



# Expand dimensions to add channel dimension for grayscale images

X_train = np.expand_dims(X_train, axis=-1)

X_val = np.expand_dims(X_val, axis=-1)



# Define CNN model architecture

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input_shape=X_train.shape[1:]),

    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(64, activation='relu'),

    Dense(len(np.unique(labels)), activation='softmax')  # Number of classes is the number of unique labels

])

# Compile the model
```

```
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])
# Define callbacks (saving the model)
checkpoint_filepath = os.path.join(output_dir, "model_checkpoint.h5")
model_checkpoint_callback = ModelCheckpoint(filepath=checkpoint_filepath,
                                            save_weights_only=False,
                                            monitor='val_accuracy',
                                            mode='max',
                                            save_best_only=True)

# Train the model
history = model.fit(X_train, tf.keras.utils.to_categorical(y_train),
                    batch_size=32,
                    epochs=10,
                    validation_data=(X_val, tf.keras.utils.to_categorical(y_val)),
                    callbacks=[model_checkpoint_callback])

# Evaluate the model
loss, accuracy = model.evaluate(X_val, tf.keras.utils.to_categorical(y_val))
print(f"Validation Loss: {loss}, Validation Accuracy: {accuracy}")

# Save the model
```

```
model.save(os.path.join(output_dir, "trained_model.h5"))

print("Model saved successfully.")

import os

import cv2

import numpy as np

from tensorflow.keras.models import load_model

from sklearn.metrics.pairwise import cosine_similarity

import matplotlib.pyplot as plt

# Function to preprocess an image

def preprocess_image(image_path, target_size=(100, 100)):

    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    if image is not None:

        image = cv2.resize(image, target_size) / 255.0  # Normalize pixel values to [0, 1]

        image = np.expand_dims(image, axis=-1)  # Add channel dimension for grayscale image

        return image

    else:

        return None

# Load trained CNN model

model_path = "/content/drive/MyDrive/trained_model/trained_model.h5"  # Replace with the path to your trained model

model = load_model(model_path)

# Function to generate embeddings for an image using the trained model

def generate_embedding(image):
```

```
    return model.predict(np.expand_dims(image, axis=0))
# Function to perform image matching
def image_matching(upload_image_path, dataset_dir, top_k=5):
    # Preprocess the uploaded image
    upload_image = preprocess_image(upload_image_path)
    if upload_image is None:
        print("Error: Unable to preprocess the uploaded image.")
        return
    # Generate embedding for the uploaded image
    upload_embedding = generate_embedding(upload_image)
    # Iterate through each image in the dataset directory
    similarity_scores = []
    for person_name in os.listdir(dataset_dir):
        person_dir = os.path.join(dataset_dir, person_name)
        for filename in os.listdir(person_dir):
            image_path = os.path.join(person_dir, filename)
            # Preprocess the dataset image
            dataset_image = preprocess_image(image_path)
            if dataset_image is None:
                print(f"Error: Unable to preprocess the dataset image {image_path}. Skipping.")
                continue
            # Generate embedding for the dataset image
            dataset_embedding = generate_embedding(dataset_image)
```

```
            # Calculate cosine similarity between the embeddings
            similarity = cosine_similarity(upload_embedding, dataset_embedding)[0][0]
            similarity_scores.append((person_name, filename, similarity))
    # Rank images based on similarity scores
    similarity_scores.sort(key=lambda x: x[2], reverse=True)
    # Return top k matches
    return similarity_scores[:top_k]
# Function to display images
def display_images(image_paths, titles):
    num_images = len(image_paths)
    fig, axes = plt.subplots(1, num_images, figsize=(12, 4))
    for i in range(num_images):
        image = cv2.imread(image_paths[i], cv2.IMREAD_GRAYSCALE)
        axes[i].imshow(image, cmap='gray')
        axes[i].set_title(titles[i])
        axes[i].axis('off')
    plt.show()
# Define paths
upload_image_path = "/content/drive/MyDrive/IMG_20231204_140203.jpg"  # Replace with the path to the uploaded image
dataset_dir = "/content/drive/MyDrive/preprocessed_dataset"  # Replace with the path to your preprocessed dataset directory
# Perform image matching
```

top_matches = image_matching(upload_image_path, dataset_dir)

Display top matched images

image_paths = [os.path.join(dataset_dir, match[0], match[1]) for match in top_matches]

titles = [f"Similarity: {match[2]:.4f}" for match in top_matches]

display_images(image_paths, titles)

GROUP MATCHING:

```
import os
import cv2
import matplotlib.pyplot as plt
def preprocess_dataset(dataset_dir, output_dir, target_size=(100, 100)):
    # Create the output directory if it doesn't exist
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    # Loop through each subdirectory (each person) in the dataset directory
    for person_name in os.listdir(dataset_dir):
        person_dir = os.path.join(dataset_dir, person_name)
        if os.path.isdir(person_dir):
            output_person_dir = os.path.join(output_dir, person_name)
            if not os.path.exists(output_person_dir):
                os.makedirs(output_person_dir)

            # Loop through each image file in the person's directory
            for filename in os.listdir(person_dir):
```

```
            image_path = os.path.join(person_dir, filename)
            output_image_path = os.path.join(output_person_dir, filename)
            # Read the image
            image = cv2.imread(image_path)
            # Preprocess the image (resize, normalize)
            if image is not None:
                image = cv2.resize(image, target_size)  # Resize to target size
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
                # Add additional preprocessing steps as needed (e.g., normalization)
                # Save the preprocessed image
                cv2.imwrite(output_image_path, image)
            else:
                print(f"Unable to read image: {image_path}")
def display_images(dataset_dir, num_images=10):
    fig, axes = plt.subplots(2, 5, figsize=(15, 6))
    axes = axes.flatten()
    images_count = 0
    # Loop through each subdirectory (each person) in the dataset directory
    for person_name in os.listdir(dataset_dir):
        person_dir = os.path.join(dataset_dir, person_name)
        if os.path.isdir(person_dir):
            # Loop through each image file in the person's directory
```

```
        for filename in os.listdir(person_dir):
            if images_count >= num_images:
                break
            image_path = os.path.join(person_dir, filename)
            # Read the image
            image = cv2.imread(image_path)
            # Display the image
            if image is not None:
                axes[images_count].imshow(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB))
                axes[images_count].set_title(person_name)
                axes[images_count].axis('off')
                images_count += 1
    plt.tight_layout()
    plt.show()
# Define input and output directories for preprocessing
dataset_dir = "/content/drive/MyDrive/MatchedImages"  # Replace with the path to
your dataset directory
output_dir = "/content/drive/MyDrive/grouppre"  # Replace with the desired output
directory
# Preprocess the dataset
preprocess_dataset(dataset_dir, output_dir)
print("Preprocessing successfully done.")
# Display 10 images from the preprocessed dataset
```

```
display_images(output_dir, num_images=10)

import os

import numpy as np

import cv2

import tensorflow as tf

from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.losses import CategoricalCrossentropy

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint

# Load preprocessed dataset

def load_dataset(dataset_dir):

    images = []

    labels = []

    for label, person_name in enumerate(os.listdir(dataset_dir)):

        person_dir = os.path.join(dataset_dir, person_name)

        for filename in os.listdir(person_dir):

            image_path = os.path.join(person_dir, filename)

            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            images.append(image)

            labels.append(label)

    return np.array(images), np.array(labels)
```

```
# Define dataset directory and output directory for the model

dataset_dir = "/content/drive/MyDrive/grouppre"  # Replace with the path to your preprocessed dataset

output_dir = "/content/drive/MyDrive/trained_model"  # Replace with the desired output directory for the trained model

# Load preprocessed dataset

images, labels = load_dataset(dataset_dir)

# Check unique labels and number of classes

unique_labels = np.unique(labels)

num_classes = len(unique_labels)

print("Unique Labels:", unique_labels)

print("Number of Classes:", num_classes)

# Ensure labels are within range

if np.max(labels) >= num_classes:

    print("Correcting labels...")

    labels = labels % num_classes  # Correct labels to be within the range of classes

# Split dataset into training and validation sets

X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)

# Normalize pixel values to [0, 1]

X_train = X_train / 255.0

X_val = X_val / 255.0

# Expand dimensions to add channel dimension for grayscale images

X_train = np.expand_dims(X_train, axis=-1)
```

```
X_val = np.expand_dims(X_val, axis=-1)

# Convert target labels to one-hot encoding

y_train_one_hot = tf.keras.utils.to_categorical(y_train, num_classes=num_classes)

y_val_one_hot = tf.keras.utils.to_categorical(y_val, num_classes=num_classes)

# Define CNN model architecture

model = Sequential([

   Conv2D(32, (3, 3), activation='relu', input_shape=X_train.shape[1:]),

   MaxPooling2D((2, 2)),

   Conv2D(64, (3, 3), activation='relu'),

   MaxPooling2D((2, 2)),

   Flatten(),

   Dense(64, activation='relu'),

   Dense(num_classes, activation='softmax')  # Number of classes is the number of unique labels

])



# Compile the model

model.compile(optimizer=Adam(),

            loss=CategoricalCrossentropy(),

            metrics=['accuracy'])

# Define callbacks (saving the model)

checkpoint_filepath = os.path.join(output_dir, "Group_model_checkpoint.h5")

model_checkpoint_callback = ModelCheckpoint(filepath=checkpoint_filepath,
```

```
                                save_weights_only=False,
                                monitor='val_accuracy',
                                mode='max',
                                save_best_only=True)
# Train the model
history = model.fit(X_train, y_train_one_hot,
                    batch_size=32,
                    epochs=10,
                    validation_data=(X_val, y_val_one_hot),
                    callbacks=[model_checkpoint_callback])
# Evaluate the model
loss, accuracy = model.evaluate(X_val, y_val_one_hot)
print(f"Validation Loss: {loss}, Validation Accuracy: {accuracy}")
# Save the model
model.save(os.path.join(output_dir, "group_trained_model.h5"))
print("Model saved successfully.")
import os
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
# Function to preprocess an image
```

```
def preprocess_image(image_path, target_size=(100, 100)):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is not None:
        image = cv2.resize(image, target_size) / 255.0  # Normalize pixel values to [0, 1]
        image = np.expand_dims(image, axis=-1)  # Add channel dimension for grayscale image
        return image
    else:
        return None
# Load trained CNN model
model_path = "/content/drive/MyDrive/trained_model/group_trained_model.h5"  # Replace with the path to your trained model
model = load_model(model_path)
# Function to generate embeddings for an image using the trained model
def generate_embedding(image):
    return model.predict(np.expand_dims(image, axis=0))
# Function to perform image matching
def image_matching(upload_image_path, dataset_dir, top_k=6):
    # Preprocess the uploaded image
    upload_image = preprocess_image(upload_image_path)
    if upload_image is None:
        print("Error: Unable to preprocess the uploaded image.")
        return
    # Generate embedding for the uploaded image
```

```
    upload_embedding = generate_embedding(upload_image)
    # Iterate through each image in the dataset directory
    similarity_scores = []
    for person_name in os.listdir(dataset_dir):
        person_dir = os.path.join(dataset_dir, person_name)
        for filename in os.listdir(person_dir):
            image_path = os.path.join(person_dir, filename)
            # Preprocess the dataset image
            dataset_image = preprocess_image(image_path)
            if dataset_image is None:
                print(f"Error: Unable to preprocess the dataset image {image_path}. Skipping.")
                continue
            # Generate embedding for the dataset image
            dataset_embedding = generate_embedding(dataset_image)
            # Calculate cosine similarity between the embeddings
            similarity = cosine_similarity(upload_embedding, dataset_embedding)[0][0]
            similarity_scores.append((person_name, filename, similarity))
    # Rank images based on similarity scores
    similarity_scores.sort(key=lambda x: x[2], reverse=True)
    # Return top k matches
    return similarity_scores[:top_k]
# Function to display images
```

```
def display_images(image_paths, titles):
    num_images = len(image_paths)
    fig, axes = plt.subplots(1, num_images, figsize=(12, 4))
    for i in range(num_images):
        image = cv2.imread(image_paths[i], cv2.IMREAD_GRAYSCALE)
        axes[i].imshow(image, cmap='gray')
        axes[i].set_title(titles[i])
        axes[i].axis('off')
    plt.show()
# Define paths
upload_image_path = "/content/drive/MyDrive/group phots/WhatsApp Image 2024-02-14 at 11.01.31_4937f73c.jpg"  # Replace with the path to the uploaded image
dataset_dir = "/content/drive/MyDrive/grouppre"  # Replace with the path to your preprocessed dataset directory
# Perform image matching
top_matches = image_matching(upload_image_path, dataset_dir)
# Display top matched images
image_paths = [os.path.join(dataset_dir, match[0], match[1]) for match in top_matches]
titles = [f"Similarity: {match[2]:.4f}" for match in top_matches]
display_images(image_paths, titles)
```

```
from flask import Flask, render_template, request
import os
import cv2
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
app = Flask(__name__)
# Function to preprocess an image
def preprocess_image(image_path, target_size=(100, 100)):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is not None:
        image = cv2.resize(image, target_size) / 255.0  # Normalize pixel values to [0, 1]
        image = np.expand_dims(image, axis=-1)  # Add channel dimension for grayscale image
        return image
    else:
        return None
# Load trained CNN model
model_path = "trained_model/trained_model.h5"  # Replace with the path to your trained model
model = load_model(model_path)
# Function to generate embeddings for an image using the trained model
def generate_embedding(image):
    return model.predict(np.expand_dims(image, axis=0))
```

```
# Function to perform image matching
def image_matching(upload_image_path, dataset_dir, top_k=5):
    # Preprocess the uploaded image
    upload_image = preprocess_image(upload_image_path)
    if upload_image is None:
        return None
    # Generate embedding for the uploaded image
    upload_embedding = generate_embedding(upload_image)
    # Iterate through each image in the dataset directory
    similarity_scores = []
    for person_name in os.listdir(dataset_dir):
        person_dir = os.path.join(dataset_dir, person_name)
        for filename in os.listdir(person_dir):
            image_path = os.path.join(person_dir, filename)
            # Preprocess the dataset image
            dataset_image = preprocess_image(image_path)
            if dataset_image is None:
                continue
            # Generate embedding for the dataset image
            dataset_embedding = generate_embedding(dataset_image)
```

```
            # Calculate cosine similarity between the embeddings
            similarity = cosine_similarity(upload_embedding, dataset_embedding)[0][0]
            similarity_scores.append((person_name, filename, similarity))
    # Rank images based on similarity scores
    similarity_scores.sort(key=lambda x: x[2], reverse=True)
    # Return top k matches
    return similarity_scores[:top_k]


# Route for the home page
@app.route('/')
def home():
    return render_template('index1.html')
# Route for image upload and matching
@app.route('/match_images', methods=['POST'])
def match_images():
    # Get the uploaded image file
    uploaded_file = request.files['file']
    if uploaded_file.filename != '':
        # Save the uploaded image
        uploaded_image_path = "uploaded_images/uploaded_image.jpg"  # Path to save the uploaded image
```

```
        uploaded_file.save(uploaded_image_path)
        # Perform image matching
        top_matches = image_matching(uploaded_image_path, "preprocessed_dataset")
        # Display top matched images
        image_paths = ["preprocessed_dataset/" + match[0] + "/" + match[1] for match in top_matches]
        titles = [f"Similarity: {match[2]:.4f}" for match in top_matches]

        return render_template('result.html', image_paths=image_paths, titles=titles)
    else:
        return "No file uploaded"
if __name__ == '__main__':
    app.run(debug=True)
```

9. RESULT ANALYSIS

OpenCV is a powerful tool for image processing and computer vision tasks, making it well-suited for developing an attendance management system based on facial recognition.

Using Flask for my interface is a good as it will allow you to create a web-based interface for your attendance management system. Flask is a lightweight and flexible framework, making it easy to integrate with my OpenCV-based backend.

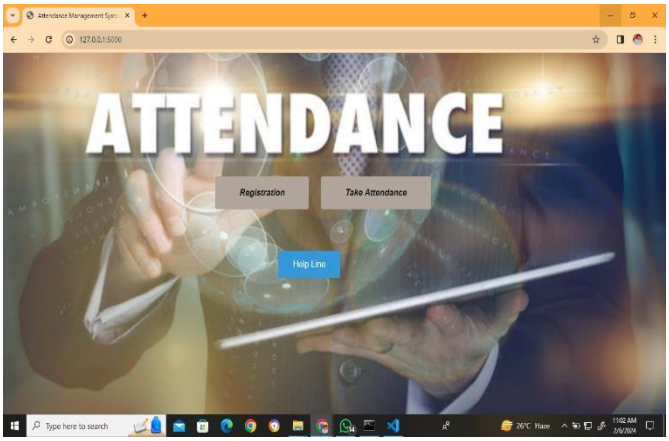

Fig 9.1: web interface

The fig 9.1 The web interface of the smart attendance system contains three main features: Registration, Attendance Taking, and Help.

Registration: This feature allows new users to register themselves in the system by providing their basic information such as name, ID, and possibly a photograph. This information is then stored in the system's database for future reference.

Attendance Taking: The core functionality of the system, this feature enables users to take attendance by capturing faces using a camera or webcam. The system then processes these faces using advanced algorithms to identify individuals and mark their attendance.

Help: This feature provides users with assistance and guidance on how to use the system effectively. It may include tutorials, FAQs, or contact information for technical support.

The web interface's user-friendly design and intuitive layout make it easy for users to navigate between these features and efficiently manage attendance records.

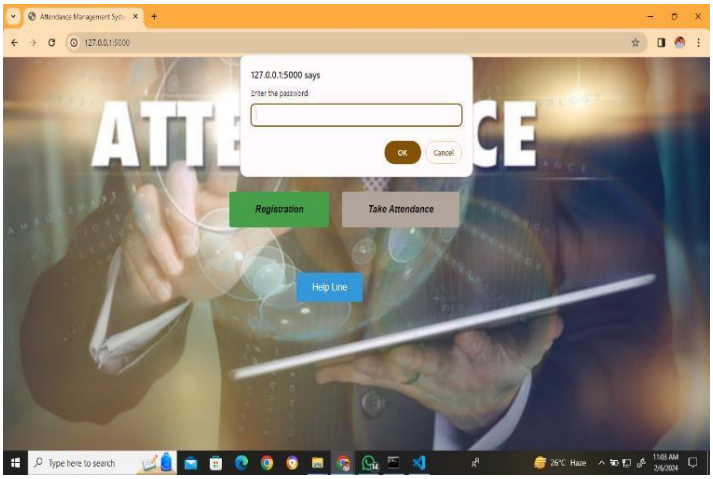

Fig 9.2: Authentication

The fig 9.2 shows the user clicks on the "Registration" button in the web interface, a popup message appears, prompting them to enter a password. This feature ensures that only authorized users, such as the admin or main person, can register new users. If the admin enters the correct password, the system navigates to the registration page, where new users can be added to the system. However, if the admin enters an incorrect password, the system displays a message indicating that the password is incorrect and prompts the admin to enter the correct password. This additional layer of security helps protect the system from unauthorized access and ensures that only authorized personnel can manage user registrations.

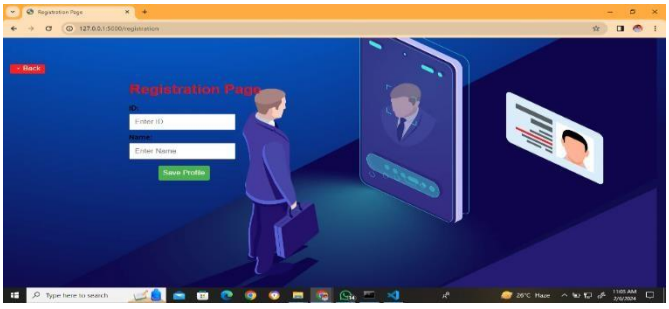

Fig 9.3: User Registration

The fig 9.3 In the registration process of the smart attendance system, users provide their names and IDs. Upon clicking the "Save Profile" button, the system opens the camera to capture several face photos in different poses and expressions. These photos are then processed using the implemented algorithms to extract facial features and create a unique profile for the user. The photos are saved in a designated folder for future reference. Finally, a message is displayed on the interface confirming that the profile has been successfully saved, ensuring that the registration process is complete and the user's information is securely stored in the system.

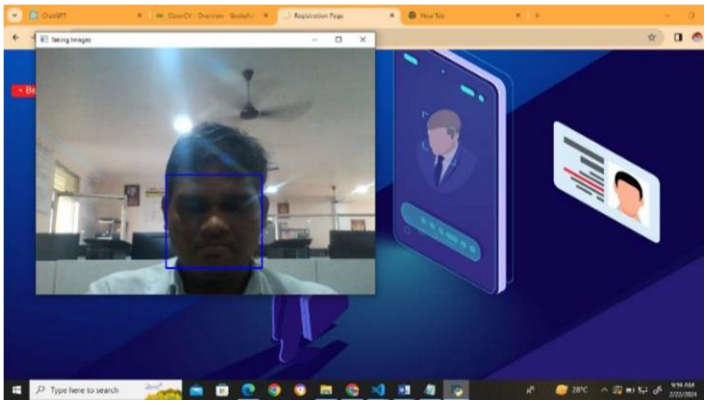

Fig 9.4 : Image’s capturing

Fig 9.4 Image’s capturing the after saving the image, the system will perform face recognition to take attendance. It will open and identify the faces, displaying the person based on our data otherwise, it will show that the face was not recognized. After recognizing faces, the camera will turn off automatically, and the recognized faces' names and IDs will be stored in one Excel sheet, including the time and date. When we click the download button, the Excel sheet will be downloaded to PC.

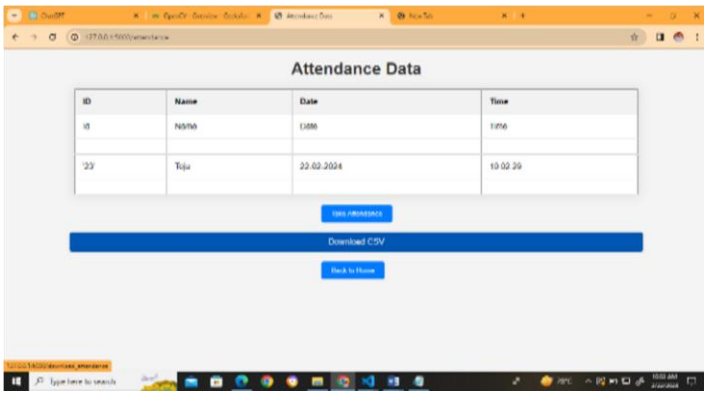

Fig 9.5: attendance record in excel sheet

Fig 9.5 capturing attendance and save attendance Fig 10 shows the process of capturing attendance and saving it in an Excel sheet. The smart attendance system captures the faces of individuals using a camera or webcam. These faces are then processed using the implemented algorithms, such as Eigenfaces and Fisherfaces, for precise detection and recognition. Once the faces are recognized, the system automatically updates the attendance records in an Excel sheet, along with timestamps for each entry. Saving attendance in an Excel sheet provides a convenient and accessible way to manage and track attendance records. It allows for easy organization and analysis of attendance data, making it simpler for administrators to monitor attendance trends and identify any irregularities. Moreover, the Excel sheet can be easily exported or integrated with other software for further processing or reporting purposes. The integration of facial recognition technology with Excel for attendance management enhances the efficiency and accuracy of attendance tracking systems, making them more reliable and userfriendly.

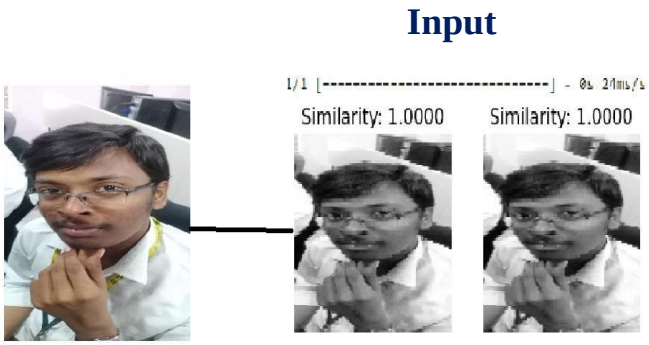

Fig 9.6: Single photo matching

Fig 9.7: Group photo matching

Fig 9.6 &9.7 shows the result of the single photo or group photo matching using the CNN algorithm is displayed in the smart attendance system's interface. When a user uploads a single photo or a group photo, the system processes the images using the Convolutional Neural Network (CNN) algorithm to identify and match faces with those in the database. For a single photo, the system compares the uploaded image with the database of registered faces and

returns the matched face along with the user's information, such as name and ID. The result is displayed on the interface, indicating whether the matching was successful or not. In the case of a group photo, the system detects and recognizes multiple faces in the image. It then matches each face with the database and displays the results individually for each recognized face. This feature allows for efficient and accurate identification of individuals in group photos, making it easier to track attendance in events or gatherings where multiple people are present.

10. Future Scope:

The proposed model has the capability of detecting and recognizing different faces and images from the camera. The face recognized matching purpose we using another module OpenCV. It's a powerful library for computer vision tasks, and it should work well for your facial recognition system. The resultant of this entire process is nothing but creating an attendance marking system in which the unique id, name and some more details of the recognized faces could be entered automatically into a CSV file. In further updating this attendance system can be taken towards web development by creating a website and marking the attendance of the people automatically into the website of the organization so that there would be no need to update or mark attendance manually. Additionally, the model's ability to find the best match photos from the dataset when uploading single or group photos enhances its utility and effectiveness. Presently this project is developed using Flask python and is successfully running on web. In the future upation, a mobile application will be developed in which each and every student are given access with unique login details so that they can track their status of attendance from anywhere round the globe.

11. Conclusion

The proposed model has the capability of detecting and recognising different faces and images from the camera. The data set which contains the images are pre-trained and tested using deep learning so that the input images would be well detected. This method is secure enough, reliable and available for use. Further CNN adds robustness to the model and using this approach of training data, 95.21% recognition rate has been achieved. Deep learning has advantage over machine learning for other face recognition techniques. The most efficient technique is the LBPH algorithm by which the problems of local features are rectified. The resultant of this entire process is nothing but creating an attendance marking system in which

the unique id, name and some more details of the recognised faces could be entered automatically into a CSV file. In further updation this attendance system can be taken towards web development by creating a website and marking the attendance of the people automatically into the website of the organisation so that there would be no need to update or mark attendance manually.

12. REFERENCES

[1] H. Yang and X. Han, "Face Recognition Attendance System Based on Real-Time Video Processing", *IEEE Access*, vol. 8, pp. 159143159150, 2020.

[2] D. M. Prasanna and C. G. Reddy, "Development of Real Time Face Recognition System Using OpenCV", *International Research Journal of Engineering and Technology*.

[3] Siti Ummi Masruroh et al., "NFC Based Mobile Attendence System with Facial Authorization on Raspberry Pi and Cloud Server", *2018 6th International Conference on Cyber and IT Service Management (CITSM)*, pp. 1-6, 2018.

[4] J. Alghamdi, R. Alharthi, R. Alghamdi, W. Alsubaie, R. Alsubaie, D. Alqahtani, et al., "A survey on face recognition algorithms", *2020 3rd International Conference on Computer Applications & Information Security (ICCAIS)*, pp. 1-5, 2020.

[5] S. M. Anzar, N. P. Subheesh, A. Panthakkan, S. Malayil and H. A. Ahmad, "Random interval attendance management system (RIAMS): A novel multimodal approach for post-COVID virtual learning", *IEEE Access*, vol. 9, pp. 91001-91016, 2021.

[6] J. W. D'Souza, S. Jothi and A. Chandrasekar, "Automated attendance marking and management system by facial recognition using histogram", *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, pp. 66-69, 2019.

[7] T. Sharanya, U. Kasturi, P. Sucharith, T. Mahesh and V. Dhivya, "Online attendance using facial recognition", *Int. J. Eng. Res*, vol. 9, no. 06, pp. 202-207, 2020.

[8] N. Yasmeen, K. K. R. C and S. Doss, "Intelligent systems powered hourly attendance capturing system", *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1536-1541, 2023.

[9] R. S, K. A, A. R and C. K. Kannan, "Automatic attendance monitoring system using LBPH and HAAR algorithm", *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, pp. 1604-1608, 2023.

[10] K. Vignesh, A. Abirami, P. Manideep, K. Vasu, K. Rakesh and S. V. Vardhan, "Smart attendance system using deep learning", *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1081-1087, 2023.

[11] NING ZHU, ZEKUAN YU and CAIXIA KOU, "A New Deep Neural Architecture Search Pipeline for Face Recognition", *IEEE Access*, 2020.

[12] ASMA BAOBAID, MAHMOUD MERIBOUT, VARUN KUMAR TIWARI and JUAN PABLO PENA, "Hardware Accelerators for Real-Time Face Recognition: A Survey", *IEEE Access*, 2022.

[13] Jian ZHAO, Chao ZHANG, Shunli ZHANG, Tingting LU, Weiwen SU and Jian JIA, "Pre-detection and binary-wordbook meager representation grounded face recognition algorithm in non-sufficient training samples", *Journal of Systems Engineering and Electronics*, 2018.

[14] ZAKIR KHAN, ARIF IQBAL UMAR, SYED HAMAD SHIRAZI, MUHAMMAD SHAHZAD, MUHAMMAD ASSAM, MUHAMMAD TAREK I. M. EL-WAKAD, et al., "Face Recognition via Multi-LeveI3D-GAN Colorization", *IEEE Access*, 2022.

[15] M. SAMI ZITOUNI, PETER LEE, UICHIN LEE, LEONTIOS I. HADJILEONTIADIS and AHSAN KHANDOKER, "Privacy Aware Affective State Recognition from Visual Data", *IEEE Access*, 2022.

[16] Ashish Yadav, Aman Sharma and Sudeept Singh Yadav, "Attendance Management System Based on Face Recognition Using Haar-Cascade", *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*.

[17] Syam Kakarla, Priyaranjan Gangula, M. Sai Rahul, C. Sai Charan Singh and T. Hitendra Sarma, "IEEE-HYDCON Smart Attendance Management System Based on Face Recognition Using CNN", *2020 IEEE-HYDCON*, 2020.