STUDENT CLASS ATTENDANCE SYSTEM USING FACE RECOGNITION.

Himethma Yapa
KIC-HNDCSAI-Y2-231-f-003

Abstract

This project develops an automated student class attendance system utilizing face recognition technology, implemented using a Convolutional Neural Network (CNN). The system captures real-time facial images via webcam and processes them for identification, logging attendance data including student names, dates, and timestamps into an Excel sheet. And store them in MongoDB. Initially, the project aimed to implement the advanced VarKFaceNet architecture, which is optimized for larger datasets and high-performance environments. However, due to the small size of the dataset, the decision was made to design a custom CNN architecture, balancing accuracy with computational efficiency for resource-constrained environments. The system achieved a recognition accuracy of 97.41% with a validation loss of 0.0721 after initial training. After augmenting the dataset to improve generalization, the model's accuracy reached 93%, reflecting the robustness gained from data augmentation techniques like random zoom, rotation, and contrast adjustments. The model incorporates regularization techniques such as L2 regularization and dropout layers to prevent overfitting and improve performance. The system not only ensures accurate and secure attendance tracking but also eliminates issues like proxy attendance and the inefficiencies of manual systems, making it a scalable solution for classroom management. This project showcases the power of deep learning in automating traditional processes and provides a foundation for future enhancements, including larger datasets and more advanced models.

Table of Content

Introduction

Attendance tracking is a fundamental part of educational institutions, ensuring accountability and participation among students. Traditional methods, manually calling roll or using attendance sheets, have long been the standard. However, these approaches are often inefficient and fraught with challenges. For instance, manual processes are time consuming, especially for large classrooms, where calling names or distributing attendance sheets can take up significant instructional time. Additionally, errors such as missed entries, duplicate records are common problems in manual attendance system.

To address these challenges, automated attendance systems using face recognition technology have emerged as a promising solution. Face recognition leverages a person's unique facial features, offering a non-intrusive, fast and reliable way to identify individuals. Unlike conventional methods requiring physical tokens like ID cards or manual interaction, face recognition systems utilize artificial intelligence, deep learning and computer vision techniques to detect, identify and verify faces in real time.

This project focuses on developing student class attendance system that uses face recognition to automate attendance management. A webcam captures real time student images which are processed by a Convolutional Neural Network (CNN) based model trained to identify their faces. One recognized, the system logs attendance including the student's name, ID, date, time and status into an Excel sheet for accurate and secure record keeping.

By leveraging CNNs and face recognition libraries, the system improves efficiency, eliminates proxy attendance, reduces administrative tasks. It provides a scalable and reliable solution for classrooms, showcasing how AI-driven automation can enhance routine processes in education.

1)Literature Review

Face Recognition Systems: An Overview

Face recognition technology has seen substantial growth in the past decade, particularly in applications requiring automated, reliable, and contactless identification methods. In security access control, attendance tracking, and identity verification, face recognition systems replace older, more error-prone systems like RFID or manual check-ins. These systems are powered by deep learning models, specifically Convolutional Neural Networks (CNNs), which have proven to be highly effective in learning intricate features from images and performing accurate classifications without the need for manual feature extraction.

The importance of using CNNs for face recognition is highlighted by their ability to automatically learn and extract spatial hierarchies of features from raw pixel data. This self-learning capability enables CNN-based systems to adapt to different conditions such as varying lighting, poses, and expressions, which are significant challenges for traditional methods. In comparison to classical computer vision techniques that rely on handcrafted algorithms like Haar Cascades or Local Binary Patterns (LBP), CNNs offer a much more scalable, robust, and accurate solution, which is why they are the preferred choice for modern face recognition tasks. CNNs allow for direct feature learning from raw data, making them suitable for real-world applications where faces appear under varying conditions.

Challenges in Face Recognition

Although CNNs offer substantial improvements, several challenges still impact the deployment of face recognition systems:

Pose Variations:

Face recognition systems can struggle when faces appear in various orientations. Models trained primarily on frontal images may not perform well when the face is viewed from the side, at an angle, or under occlusions. According to recent advancements in the literature, pose-invariant techniques like multi-angle data augmentation and 3D face recognition are effective in overcoming these challenges by ensuring the model is exposed to diverse viewing angles during training.

Lightning Conditions:

Lighting variations pose a significant challenge for face recognition systems. Changes in illumination can obscure critical facial features, thus impairing model performance. Techniques like Histogram Equalization and data augmentation that simulate various lighting conditions are commonly used to enhance the robustness of CNN models. By training on images with artificially altered lighting, models can generalize better under varying environmental conditions.

Facial Expressions

Facial expressions (smiling, frowning, etc.) alter the geometry of facial features. This makes it harder for face recognition systems to distinguish individuals, as the same person may appear quite different depending on their expression. According to FaceNet and similar research, the best models integrate expression-invariant features by training on diverse datasets that include multiple expressions. These techniques allow the network to focus on the most important identity-defining features while disregarding changes caused by facial expressions.

Data Scarcity

One of the most critical challenges in face recognition is the lack of large, labeled datasets. Small datasets often result in overfitting, where the model memorizes specific features and fails to generalize well to unseen data. As noted in the research, the use of data augmentation, transfer learning, and synthetic data generation techniques can effectively address the issue of limited data. Data augmentation involves generating new images by applying transformations such as flipping, rotation, and color shifting. This expands the diversity of the dataset and helps the model learn more robust and generalizable features.

Occlusions

Objects like glasses, hats, or masks often obstruct parts of the face, reducing the effectiveness of the recognition system. According to the research, models that leverage attention mechanisms and multi-view approaches can compensate for occlusions by focusing on the visible regions of the face and using alternative angles to complete recognition.

Convolutional Neural Networks (CNNs)

CNNs have become the backbone of modern face recognition systems due to their unique ability to process visual data and automatically learn hierarchical patterns in images. The use of CNNs in face recognition is essential because they enable the system to detect and classify fine-grained details such as the unique contours of the eyes, nose, and mouth, which are critical for distinguishing between different faces.

The fundamental components of a CNN that contribute to its success in face recognition include:

Convolutional Layers:

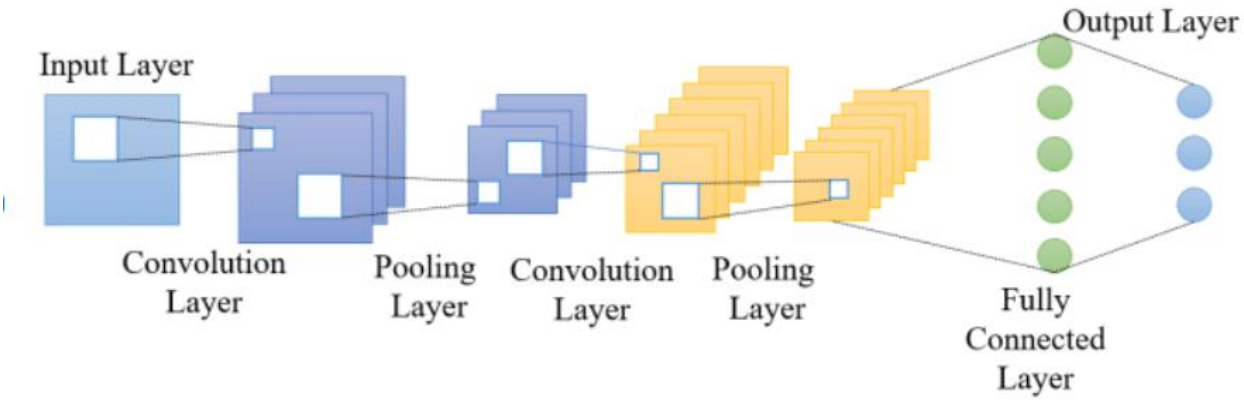
Convolutional layers are designed to automatically learn filters (or kernels) that detect simple features such as edges, textures, or patterns in the early stages, and more complex structures, such as facial contours, in deeper layers. For example, in the case of face recognition, these layers capture key features like the positioning of eyes, the curvature of the nose, and the shape of the mouth, all of which are critical for distinguishing individuals. This feature-learning ability makes CNNs particularly effective for face recognition, as they can learn subtle and complex representations that manual techniques cannot achieve.

Pooling Layers:

Pooling layers help to downsample the feature maps, reducing their dimensionality while retaining the most important information. The most common form of pooling is max pooling, which selects the maximum value from a set of neighboring pixels. This makes the model more invariant to small distortions or translations, ensuring that minor changes in facial position or expression do not significantly impact the model’s ability to recognize faces.

Fully connected Layers

Once the convolutional and pooling layers have extracted relevant features, fully connected layers combine these features and perform classification. The final output layer typically produces a probability distribution over all possible identities, deciding which individual the input image corresponds to. The fully connected layers integrate the learned features from all previous layers to output a decision, making the final classification possible.



Activation Functions:

Activation functions like ReLU (Rectified Linear Unit) introduce non-linearity into the network, allowing the model to learn complex patterns. The choice of activation function directly influences the model's capacity to learn representations that capture critical features in the data. The Softmax function is mainly used in the output layer of classification models to normalize raw logits (outputs) into probabilities, making them interpretable. It ensures the outputs of the network sum to 1, which is suitable for multi-class classification problems.

Various CNN architectures have been developed to improve the accuracy and efficiency of face recognition systems.

VGG-Face:

The VGG-Face model, based on the VGG architecture, uses a deep stack of 3x3 convolutional layers, capturing detailed facial features effectively. However, its large number of parameters (over 138 million) makes it computationally expensive, limiting its suitability for real-time applications or embedded systems.

ResNet:

ResNet introduced residual connections to mitigate the vanishing gradient problem and allow very deep networks to be trained. This architecture is highly effective for complex face recognition tasks, where accurate identification across large datasets is essential. However, the deep nature of ResNet requires substantial computational resources, making it less feasible for embedded devices.

FaceNet:

FaceNet is an innovative architecture that learns facial embeddings rather than directly classifying images. By using a triplet loss function, FaceNet minimizes the distance between embeddings of the same person while maximizing the distance between different individuals. This allows for highly efficient face verification and clustering, making it ideal for identity matching tasks.

Lightweight CNNs for Embedded Systems

Given the growing demand for real-time face recognition in resource-constrained environments, lightweight CNN architectures have been developed to balance performance and efficiency. These models are designed to run on devices with limited computational power while still maintaining high accuracy.

MobileNet:

MobileNet employs depthwise separable convolutions, which significantly reduce the number of parameters and computations. The architecture is highly efficient, allowing for real-time face recognition on mobile and embedded devices. Further optimizations in MobileNetV2 and MobileNetV3 enhance the model’s performance with inverted residual blocks.

ShuffleNet:

ShuffleNet uses group convolutions and channel shuffling to improve computational efficiency without sacrificing accuracy. This makes it ideal for low-power devices requiring real-time processing, ensuring the system remains efficient even when deployed on constrained hardware.

VarkFaceNet:

VarKFaceNet is a breakthrough in lightweight face recognition, designed specifically for embedded systems. By using variable kernel convolutions (VarKNet), the architecture dynamically adjusts kernel sizes based on the network's depth, improving the receptive field and capturing more detailed features. This approach reduces the computational cost while maintaining high accuracy, with the model achieving 99.5% accuracy on the LFW dataset.

According to the research paper, VarKFaceNet achieves remarkable results with minimal computational overhead (0.24 GFLOPs and 0.7 million parameters). This makes VarKFaceNet ideal for applications like attendance tracking, where high performance and low latency are critical. Although the project couldn't implement VarKFaceNet due to dataset constraints, its principles have guided the architecture choices, and future work will incorporate VarKFaceNet or similar models to enhance accuracy and efficiency.

Data Scarcity and Augmentation Techniques

One of the primary challenges in this project is small datasets, which can lead to overfitting and poor generalization. To address this, data augmentation techniques are employed to artificially increase the dataset size:

- **Flipping** simulates different facial orientations.
- **Rotation** allows for variations in head tilt.
- **Scaling** simulates faces of varying sizes.
- **Color Shifting** adjusts lighting and contrast conditions.

These techniques enhance the model's ability to recognize faces under diverse scenarios, improving its robustness and generalization.

Real-Time Performance and Latency

For applications like attendance tracking and identity verification, real-time performance is critical. CNNs, while powerful, can be computationally intensive, which may lead to high inference times. To address this, lightweight architectures like VarKFaceNet and MobileNet are employed, offering a good balance between accuracy and speed. Additionally, techniques like quantization, pruning, and model re-parameterization are explored to optimize model performance, ensuring that the system can run on devices with limited resources.

Privacy and Security Concerns

With the widespread deployment of face recognition systems, privacy and security concerns become increasingly important. Given the sensitive nature of facial data, it is essential to implement systems that are secure and ethically responsible.

1. Adversarial Attacks: Face recognition models are vulnerable to small, imperceptible changes in the input image, which can lead to incorrect classifications. Adversarial training techniques can help improve the robustness of the system to these attacks.

2. Data Privacy: Storing facial data poses significant privacy risks. Federated learning techniques can mitigate these risks by processing data locally, reducing the risk of breaches.

3. Ethical Concerns: Ensuring fairness and transparency is crucial. Using diverse datasets during training helps mitigate biases, ensuring the system provides equitable results across different demographics.

Model Selection and Implementation

In the development of an efficient face recognition system, selecting an appropriate model architecture is critical. Initially, the VarkFaceNet architecture was considered for implementation due to its robust performance in face recognition tasks and its ability to leverage deep feature extraction for high accuracy. VarkFaceNet is particularly renowned for its use of optimized layers and deep convolutional techniques tailored for face verification and recognition tasks. However, the suitability of any model depends significantly on the availability and size of the dataset.

Given the constraints of a small dataset, adopting the VarkFaceNet architecture posed challenges, primarily due to its reliance on extensive training data to achieve optimal performance. Training such deep models on limited data can lead to overfitting, where the model memorizes the training data instead of generalizing well to new samples.

To address these limitations, a simpler Convolutional Neural Network (CNN) architecture was adopted. CNNs are highly effective for image-based tasks, even with relatively small datasets, due to their ability to extract spatial hierarchies of features through convolutional layers. By designing a custom CNN architecture, the project leveraged fewer parameters and avoided the pitfalls of overfitting while maintaining sufficient model complexity to achieve reliable face recognition.

This pragmatic choice highlights the importance of aligning model selection with data availability and computational resources, ensuring the system's effectiveness in real-world applications. Future improvements could include augmenting the dataset or employing transfer learning to integrate pre-trained weights from larger models like VarkFaceNet.

Future Directions

Several directions can be pursued to enhance the face recognition system developed in this project:

1. Integration of Advanced Architectures: As datasets grow, integrating models like VarKFaceNet will significantly improve performance.

2. Attention Mechanisms: Using attention mechanisms will help the model focus on critical facial regions, enhancing accuracy.

3. Hybrid Models: Combining CNNs with models like Vision Transformers (ViTs) could lead to better handling of pose variations and occlusions.

4. Real-World Testing: Deploying the system in diverse real-world environments will provide insights into areas of improvement.

3)Methodology

The methodology section of this project provides a detailed outline of the processes and techniques used to develop the Face Recognition Attendance System, focusing on the automation of attendance marking through facial recognition technology. This system is designed to eliminate the manual effort involved in attendance tracking by automatically identifying individuals based on facial features. The methodology involves several key stages: data collection, preprocessing, model development, and integration into an attendance logging system. The first step includes gathering a set of facial images for training the model. These images are then preprocessed to ensure consistency and improve model accuracy. A Convolutional Neural Network (CNN) is trained to recognize distinct facial features, which are used for identification. The trained model is then deployed for real-time face recognition, and once an individual's identity is confirmed, their attendance is recorded automatically, typically into an Excel sheet and stores in mongodb. This approach streamlines the attendance process, making it faster, more accurate, and resistant to human errors or fraud, offering an efficient solution for environments such as educational institutions, offices, or events.

3.1Data collection and Preprocessing

Data collection plays a pivotal role in the success of any machine learning model, particularly in tasks like face recognition, where the quality and diversity of the dataset significantly impact model performance. In this project, the dataset was carefully structured to facilitate the identification of individuals for attendance purposes.

The dataset consists of 930 images across seven classes, organized in a folder-based structure. Each subfolder represents a unique individual (e.g., a student), labeled by their name, and contains multiple images of their face. These images capture various angles, facial expressions, and lighting conditions, ensuring that the dataset reflects a wide range of real-world scenarios.

Data Preprocessing

Before training the model, it is crucial to preprocess the collected data to ensure consistency, improve the model's performance, and mitigate the risk of overfitting. The preprocessing pipeline included image resizing, normalization, and dataset preparation.

1. Resizing Images to a Consistent Size

Images collected for training may vary in dimensions, which can lead to inconsistencies during model training. To address this, all images were resized to a uniform size of 180x180 pixels using OpenCV's cv2.resize function. This ensures that the input dimensions to the CNN model remain consistent. The resizing process also helps reduce computational complexity while preserving the key features required for face recognition.

```
X,y = [],[]

for face_name, images in face_images_dict.items():
  for image in images:
    img = cv2.imread(str(image))
    resized_img = cv2.resize(img,(180,180))
    X.append(resized_img)
    y.append(face_labels_dict[face_name])
```

2. *Normalizing Pixel Values*

To enhance the model's ability to learn effectively, pixel values of the images were normalized by dividing them by 255.0, scaling them to a range of 0 to 1. Normalization helps improve numerical stability during training and ensures that the input data is within a consistent range, making it easier for the model to converge.

```
# Normalize the datasets
X_train_scaled = X_train / 255.0
X_val_scaled = X_val / 255.0
X_test_scaled = X_test / 255.0

# Verify the shapes
print(f"X_train shape: {X_train_scaled.shape}")
print(f"X_val shape: {X_val_scaled.shape}")
print(f"X_test shape: {X_test_scaled.shape}")
```

Data Splitting

Data splitting is a crucial step in preparing the dataset for training, validation and testing, ensuring that the model is evaluated on unseen data to assess its generalization ability. Firstly the dataset is split into two parts training + validation 85% and testing 15%.

Training Set: the training set is used to train the model. It contains 70% of total dataset, selected from the training and validation subset. This set is used to update the model's weight during training to learn patterns from the data.

Testing Set: the testing set presents 15% of the dataset and this set use for final evaluation. It is used after the model has been trained to assess its performance on data that it has never see before, simulating real-world usage.

Validation Set: the validation set, which makes up 15% of the total dataset, is used during training to monitor the model's performance and fine tune hyperparameters. It helps in preventing overfitting by providing feedback on how well the model generalizes to unseen data while training.

This two-step split ensures that the training, validation, and testing data are kept separate for reliable evaluation and model tuning.

```
from sklearn.model_selection import train_test_split

# Split the dataset: 85% for training+validation and 15% for testing
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y,
test_size=0.15, random_state=0)

# Split the training+validation set: 70% training and 15% validation
X_train, X_val, y_train, y_val = train_test_split(X_train_val,
y_train_val, test_size=0.1765, random_state=0)
```

3.2) CNN Model Development

Model Architecture

This model architecture is a Convolutional Neural Network (CNN) designed for multi-class classification, built using TensorFlow's Keras API.

1. Input Layer

Description: The input layer specifies the shape of the input data: (180, 180, 3). This corresponds to images with dimensions of 180x180 pixels and three color channels (Red, Green, Blue).

Mathematical Concept: Each pixel in the input image is represented as a value between 0 and 1 (after normalization). For an image, this results in a tensor of shape (180, 180, 3). The model processes the image as a 3D grid of pixel intensities.

First convolutional block

Conv2D Layer

Description: This layer applies 16 convolutional filters of size 3x3 to the input image with 'same' padding. 'Same' padding ensures the spatial dimensions (180x180) are preserved after the convolution.

Mathematical Concept:

Convolution involves sliding a filter (kernel) over the input image and computing the dot product between the filter and local patches of the image.

For each filter:

$$Output[i,j]=k=1\sum 3l=1\sum 3Input[i+k,j+l]\cdot Filter[k,l]$$

where k and l iterate over the filter's height and width.

The result is a feature map (highlighting edges, textures, or patterns). With 16 filters, this layer produces 16 feature maps of size 180×180180 \times 180180×180.

ReLU Activation

Description: ReLU (Rectified Linear Unit) applies the non-linearity

$$f(x)=max(0,x)$$

Purpose: Introduces non-linearity, allowing the network to learn complex patterns.

MaxPooling2D Layer:

Description: A pooling layer performs downsampling by taking the maximum value in a sliding window (typically 2x2).

Mathematical Concept:

For each 2×22 region in the input, the output is:

Output[i,j]=max(Input[2i,2j],Input[2i+1,2j],Input[2i,2j+1],Input[2i+1,2j+1])

After pooling, spatial dimensions are reduced by half, resulting in feature maps of size (90,90,16)

Second Convolutional Block

Conv2D Layer

Description: This layer doubles the number of filters to 32, capturing more complex features (e.g., corners, shapes).

Mathematical Concept: The convolution operation is similar to the first block but applied to the 90×90×16 feature maps, resulting in 90×90×32 feature maps.

MaxPooling2D Layer

Description: Another pooling operation reduces spatial dimensions further to 45×45×32 retaining the most critical features while reducing computation.

Third Convolutional Block

Conv2D Layer

Description: The filter count increases to 64 to extract high-level, abstract features.

Mathematical Concept: With same padding, the output shape remains 45×45×64. These high-level features may correspond to distinct regions of a face, such as eyes, nose, or mouth.

MaxPooling2D Layer

Description: Downsamples the feature maps to 22×22×64, further reducing the spatial dimensions while retaining key information.

Flatten Layer

Description: The 3D output from the previous layers (22×22×64) is flattened into a 1D vector.

Mathematical Concept: The flattening operation converts the 3D tensor into a vector of length: 22×22×64=30,976.

Fully Connected Layers

Dense Layer:

A layer with 128 neurons and ReLU activation processes the flattened features, learning complex combinations of patterns.

$$z_i = \sum_{j=1}^{n} w_{ij} x_j + b_i, \quad \text{for } i = 1, 2, \ldots, 128$$

Purpose: Learns combinations of high-level features to map to the output classes.

Output Dense Layer:

This layer has neurons, corresponding to the number of classes in the dataset.

No activation function is specified, as the logits are passed directly to the loss function.

Mathematical Concept:

- Each neuron computes

$$\text{logit}_k = \sum_{j=1}^{n} w_{kj} x_j + b_k$$

These logits are passed to the loss function for classification.

7. Compilation

Optimizer: The Adam optimizer is used for training, combining the benefits of RMSProp and SGD for adaptive learning rates.

Combines momentum (past gradients) and adaptive learning rates:

The weights are updated using:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

Loss Function: SparseCategoricalCrossentropy is used for multi-class classification tasks where the labels are integers.

$$\text{Loss} = -\sum_{i=1}^{C} y_i \log(p_i)$$

Metrics: Accuracy is used to evaluate the model's performance during training and testing.

```
import tensorflow as tf
from tensorflow.keras import Sequential, layers

num_classes = 7
model = Sequential([
    layers.Conv2D(16, 3, padding='same', activation='relu',
input_shape=(180, 180, 3)),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])


model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logi
ts=True),
              metrics=['accuracy'])
```

Model Summary

The model.summary() function provides a concise overview of the neural network's architecture. It details each layer, including its type, output shape, and the number of trainable parameters.

The model is designed to extract hierarchical features from input images:

16 filters → 32 filters → 64 filters: Gradually increasing the filters captures patterns from simple to complex (edges → textures → facial features).

Pooling Layers: Reduce computational load and focus on essential features.

Fully Connected Layers: Aggregate features and map them to the final output classes.

This structured approach ensures effective learning, even with a relatively small dataset.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3,965,056
dense_1 (Dense)	(None, 7)	903

Total params: 3,989,543 (15.22 MB)
Trainable params: 3,989,543 (15.22 MB)
Non-trainable params: 0 (0.00 B)

I have got 0.9741 of accuracy and 0.0721 loss validation. To get a more accurate model I used data augmentation and retrained the model.

Data Augmentation

Data augmentation enhances dataset diversity by applying random transformations, improving model generalization and robustness. In this project, keras.Sequential implements:

Random Zoom: Zooms into images by up to 50%, simulating distance variations.

Random Contrast: Adjusts contrast by 10% to mimic different lighting conditions.

Random Rotation: Rotates images up to 20%, accounting for tilted angles.

Random Flip: Horizontally flips images for mirrored variability.

The input shape (180, 180, 3) matches image dimensions (height, width, and RGB channels), ensuring consistency.

```
from tensorflow import keras
from tensorflow.keras import layers

# Define image dimensions
image_height = 180
image_width = 180

data_augmentation = keras.Sequential([
    layers.Input(shape=(180, 180, 3)),
    layers.RandomZoom(0.5),
    layers.RandomContrast(0.1),
    layers.RandomRotation(0.2),
    layers.RandomFlip("horizontal",
                      input_shape=(image_height,
                                   image_width,
                                   3))
])
```

After augmenting images, I retrained the model using a updated CNN architecture.

Adjusted CNN Model Architecture

To enhance the performance of the face recognition system, an improved Convolutional Neural Network(CNN) architecture was developed, integrating advanced regularization techniques and ***data augmentation for better generalization.***

Input Layer: the model accepts input dimensions (180,180,3) (height, width and RGB channels).

Each image is represented as a tensor of shape:

Input Tensor Shape: (180,180,3)

Where 180x180 represents the spatial resolution, and 3corresponds to the RGB color channels.

Data Augmentation Layer: A data augmentation pipeline applies transformations like random zoom, contrast adjustments, rotations, and horizontal flips to artificially expand the dataset and improve robustness to variations lighting, angles, and image alignment.

Convolutional and pooling layers:

A Conv2D layer with 16 filters, a kernel size of 3x3, and ReLU activation function extracts initial features, followed by maxpooling2D for spatial downsampling. L2 regularization (weight decay) of 0.005 reduces overfitting.

$$\text{Feature Map}[i,j] = \sum_{k=1}^{3}\sum_{l=1}^{3} \text{Input}[i+k, j+l] \cdot \text{Filter}[k,l]$$

Here, a 33×3 filter slides over the input tensor to extract spatial features like edges and textures.

- Filters: 16 filters are applied.
- Output Tensor Shape: (180,180,16) (with same padding).

ReLU Activation: Applies the non-linear activation:

$$f(x)=\max(0,x)$$

This introduces non-linearity, enabling the model to learn complex patterns.

MaxPooling Operation: Reduces spatial dimensions by taking the maximum value within each 22×2 window:

Output Tensor Shape:

(90,90,16)

Second Convolutional Block:

A Conv2D layer with 32 filters, followed by max-pooling, further extracts intermediate features with the same regularization strategy.

Convolutional Output Shape:

(90,90,32)

MaxPooling Output Shape:

(45,45,32)

Third Convolutional Block

A Conv2D layer with 64 filters processes high level features, followed by max pooling for dimensionality reduction.

Convolutional Output Shape:

(45,45,64)

MaxPooling Output Shape:

(22,22,64)

Dropout Layer

A 20% dropout is applied to prevent overfitting by randomly deactivating neurons during training.

$$z_i' = \begin{cases} z_i & \text{with probability } 1 - p \\ 0 & \text{with probability } p \end{cases}$$

where $p = 0.2$ is the dropout rate.

Flatten Layer

Description: Converts the 3D tensor from the previous layer into a 1D vector.

Input shape:

(22,22,64) = 30,976 features

Output shape:

(30,976)

Fully Connected Layers

$$z_i = \sum_{j=1}^{n} w_{ij} \cdot x_j + b_i$$

Dense layer: A fully connected layer with 128 neurons and ReLU activation learns high level abstract features, regularized with L2 (0.002).

Dropout: Another 20% dropout is applied for regularization.

Output Layer: A final dense layer with softmax activation outputs class probabilities for seven categories of faces.

Compilation and optimization:

Optimizer : the Adam optimizer is configured with a higher learning rate of 0.001 to speed up learning.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Loss Function: sparse categorical cross-entropy is used to handle multiclass classification effectively.

$$\text{Loss} = -\sum_{i=1}^{C} y_i \log(p_i)$$

Metrics: Model performance is monitored using accuracy

```
import tensorflow as tf
from tensorflow.keras import layers, Sequential, regularizers
# Adjusted Model
model = Sequential([
    layers.Input(shape=(180, 180, 3)),
    data_augmentation,  # Data augmentation layer for better
generalization

    layers.Conv2D(16, 3, padding='same', activation='relu',
kernel_regularizer=regularizers.l2(0.005)),
    layers.MaxPooling2D(),

    layers.Conv2D(32, 3, padding='same', activation='relu',
kernel_regularizer=regularizers.l2(0.005)),
    layers.MaxPooling2D(),

    layers.Conv2D(64, 3, padding='same', activation='relu',
kernel_regularizer=regularizers.l2(0.005)),
    layers.MaxPooling2D(),

    layers.Dropout(0.2),  # Reduced dropout rate for better learning
```

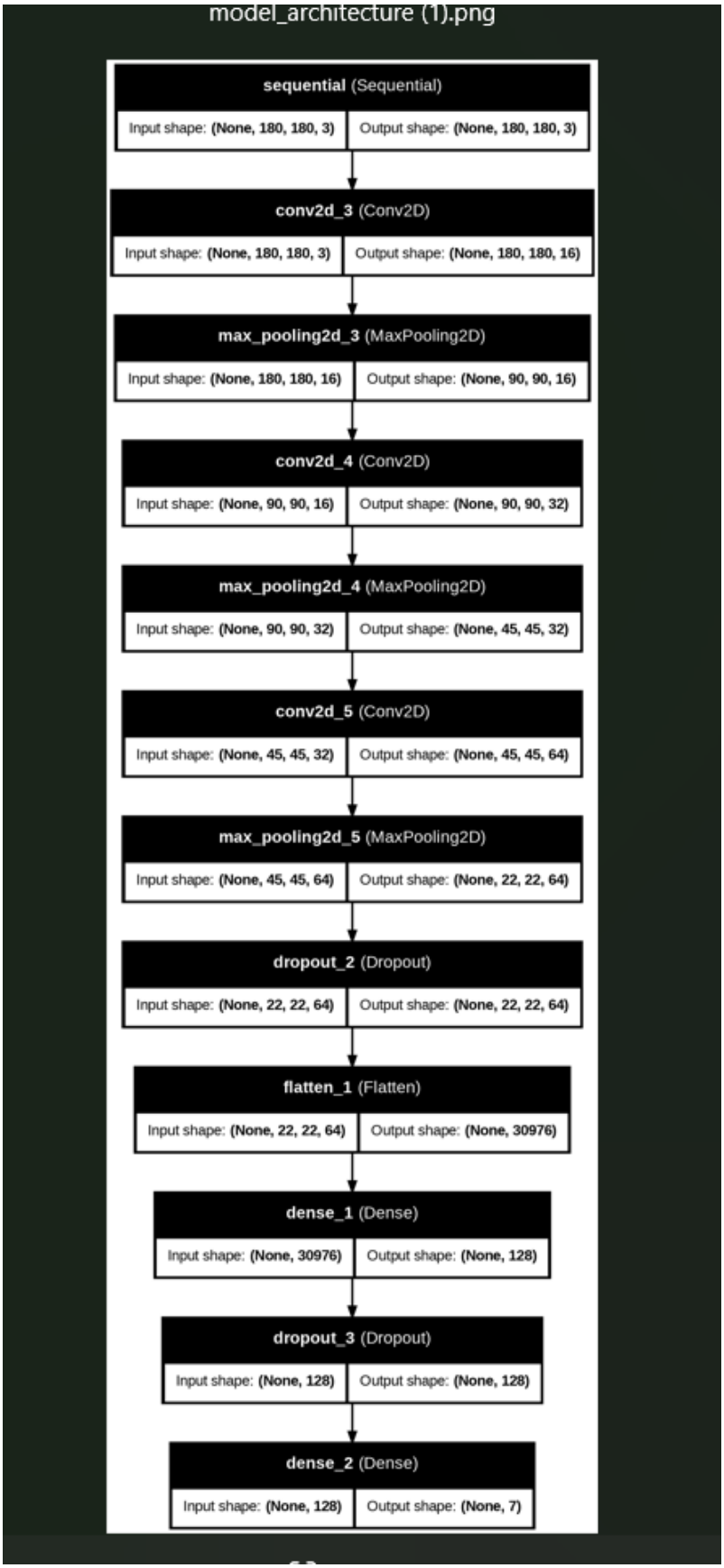
```
    layers.Flatten(),

    layers.Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.002)),
    layers.Dropout(0.2),  # Reduced dropout rate for better learning

    layers.Dense(num_classes, activation='softmax')
])

# Compile the model with a higher learning rate
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),  # Increased
learning rate
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

# Model Summary
model.summary()
```


model_architecture (1).png
sequential (Sequential)
Input shape: (None, 180, 180, 3)
Output shape: (None, 180, 180, 3)
conv2d_3 (Conv2D)
Input shape: (None, 180, 180, 3)
Output shape: (None, 180, 180, 16)
max_pooling2d_3 (MaxPooling2D)
Input shape: (None, 180, 180, 16)
Output shape: (None, 90, 90, 16)
conv2d_4 (Conv2D)
Input shape: (None, 90, 90, 16)
Output shape: (None, 90, 90, 32)
max_pooling2d_4 (MaxPooling2D)
Input shape: (None, 90, 90, 32)
Output shape: (None, 45, 45, 32)
conv2d_5 (Conv2D)
Input shape: (None, 45, 45, 32)
Output shape: (None, 45, 45, 64)
max_pooling2d_5 (MaxPooling2D)
Input shape: (None, 45, 45, 64)
Output shape: (None, 22, 22, 64)
dropout_2 (Dropout)
Input shape: (None, 22, 22, 64)
Output shape: (None, 22, 22, 64)
flatten_1 (Flatten)
Input shape: (None, 22, 22, 64)
Output shape: (None, 30976)
dense_1 (Dense)
Input shape: (None, 30976)
Output shape: (None, 128)
dropout_3 (Dropout)
Input shape: (None, 128)
Output shape: (None, 128)
dense_2 (Dense)
Input shape: (None, 128)
Output shape: (None, 7)

Model Summary

Model summary helps analyze the model's structure, understand its complexity, and ensure it matches the design goals.

Model: "sequential_6"

Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 180, 180, 3)	0
conv2d_15 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_15 (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_16 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_16 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_17 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_17 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout_8 (Dropout)	(None, 22, 22, 64)	0
flatten_5 (Flatten)	(None, 30976)	0
dense_10 (Dense)	(None, 128)	3,965,056
dropout_9 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 7)	903

Total params: 3,989,543 (15.22 MB)
Trainable params: 3,989,543 (15.22 MB)
Non-trainable params: 0 (0.00 B)

Training the Model with Advanced Callbacks

The training process of the model incorporates callbacks and customized learning rate schedular to optimize performance and prevent overfitting.

Learning Rate Schedular

The LearningRateScheduler adjusts the learning rate dynamically during training based on number of epochs.For the first 50 epochs, learning rate remains constant. Between 50 and 75 epochs, the learning rate is reduced by half to encourage fine-tuning. After 75 epochs, the learning rate is further reduced to one-tenth, ensuring gradual convergence. This helps the model

adapt better during training by accelerating learning initially and slowing down to refine its performance in later stages.

Batch Size:

The batch size of 32 specifies the number of samples processed before updating the model's parameters. This balances memory efficiency and training stability.

Validation Data:

Validation data is used to evaluate the model's performance on unseen data after each epoch. It helps monitor overfitting and ensures the model generalizes well.

Training Process:

The model is trained for 100 epochs, allowing sufficient time for optimization.

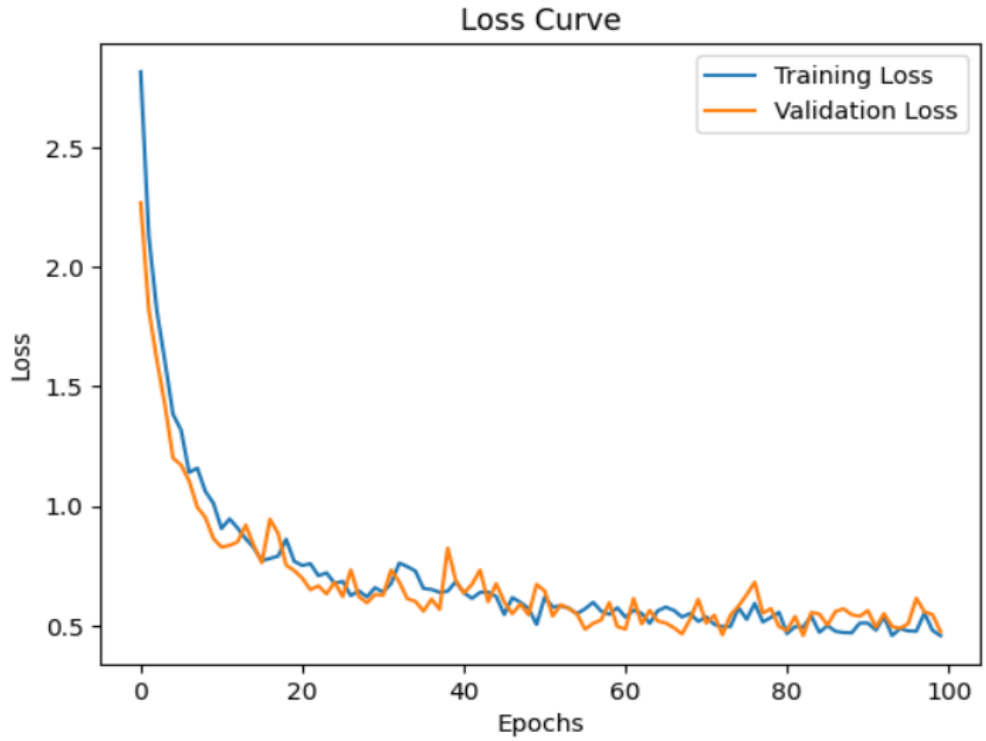
The history object stores metrics such as training and validation loss/accuracy across epochs, which can be used for later analysis (e.g., plotting learning curves).

Training and Validation Loss Curve

The loss curve displays training loss (blue line) and validation loss (orange line) across training epochs.

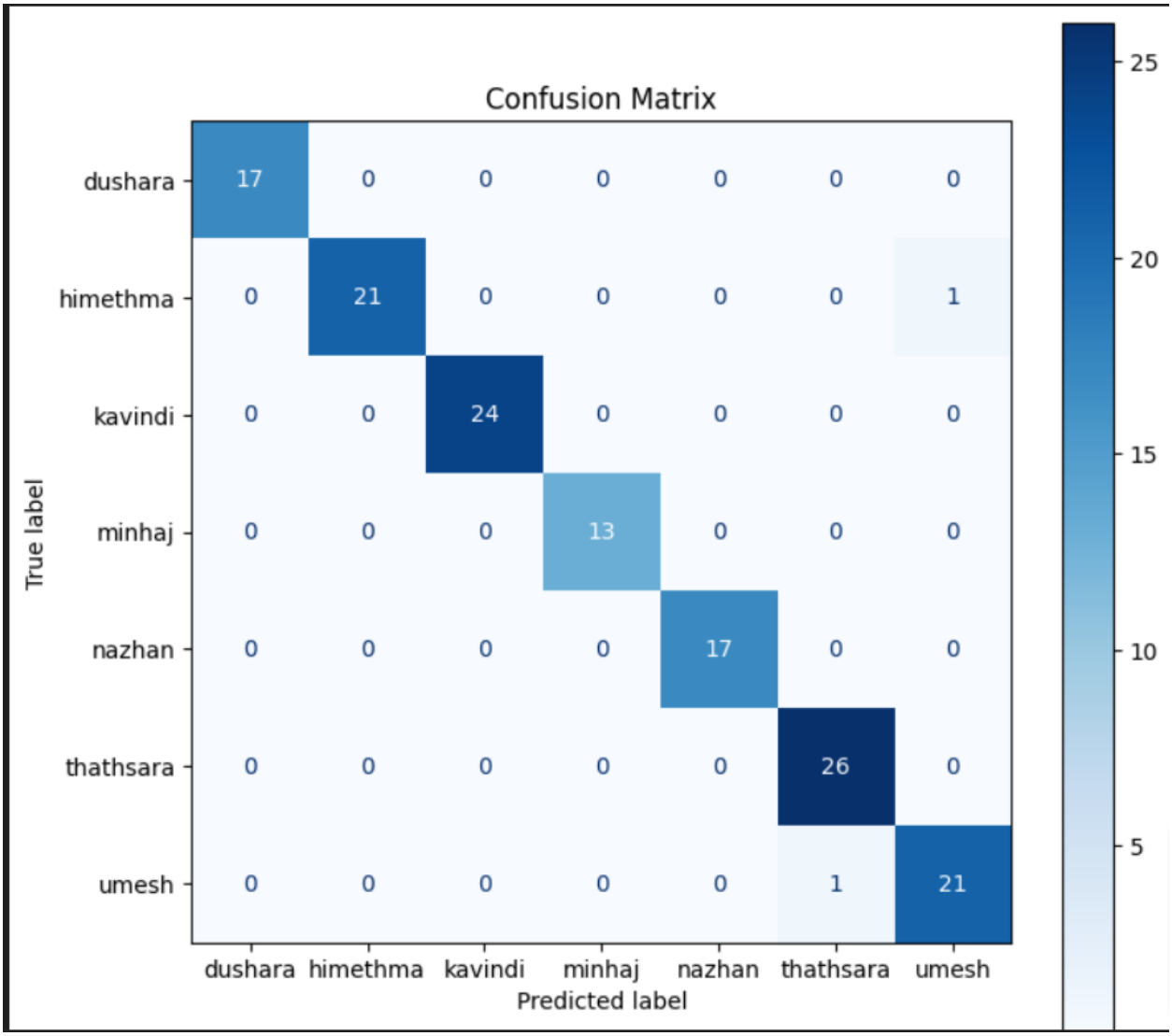
Both losses decrease rapidly during the initial epochs, showing that the model is learning effectively.

Over time, the loss values stabilize, indicating that the model is approaching convergence.



Confusion Matrix

The confusion matrix further highlights the model's classification performance, identifying areas of strong accuracy and potential misclassifications. Precision, recall, and F1-scores were calculated, demonstrating the system's robustness across various conditions.

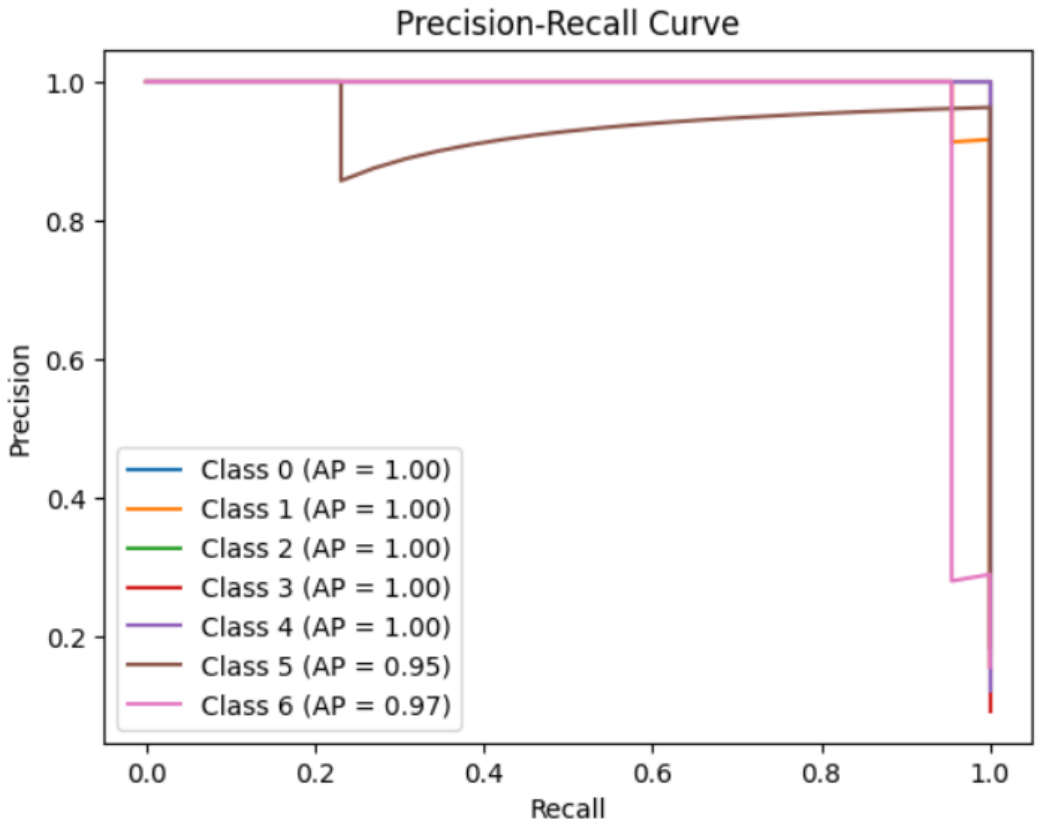


Precision-recall Curve

A Precision-Recall (PR) Curve is a visual representation of the trade-off between precision and recall for different thresholds in a classification model. It is particularly useful when dealing with imbalanced datasets, as it focuses on the performance of the positive class.

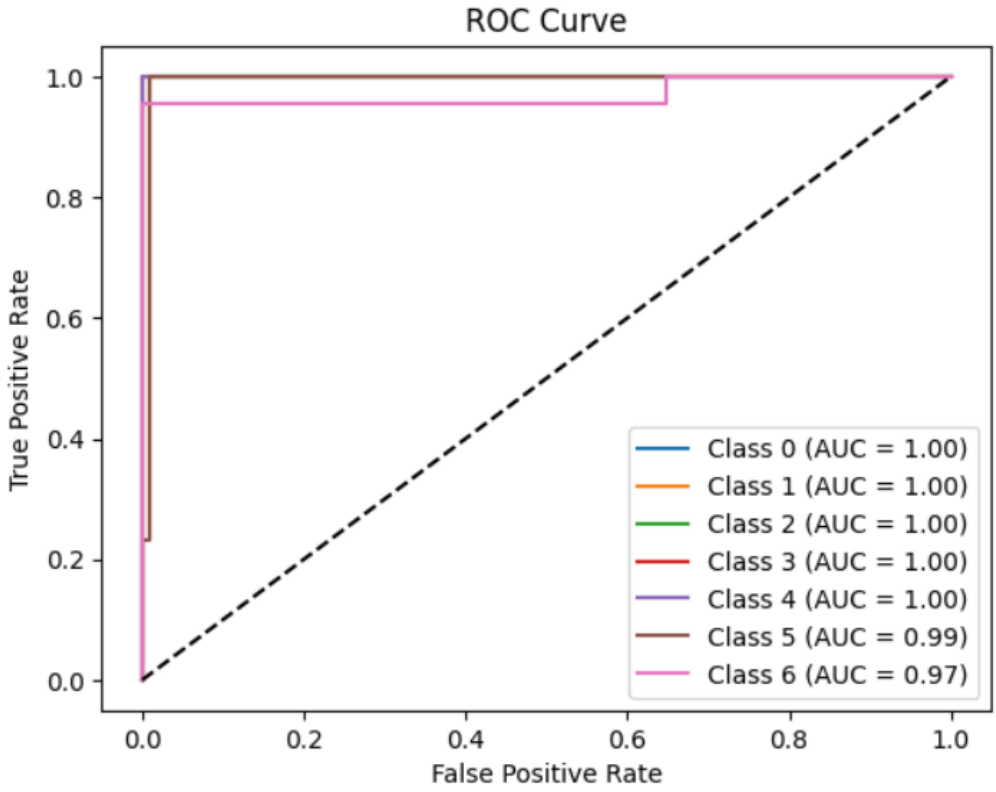
$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$



ROC Curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation of your model's performance at various classification thresholds.



Precision, Recall, and F1-Score:

Precision: Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It reflects the model's ability to avoid false positives.

Recall: Recall measures the proportion of actual positives that were correctly identified by the model. It reflects the model's ability to capture all relevant instances.

F1-Score: The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. It is particularly useful in scenarios with imbalanced classes.

	precision	recall	f1-score	support
dushara	1.00	1.00	1.00	17
himethma	1.00	0.95	0.98	22
kavindi	1.00	1.00	1.00	24
minhaj	1.00	1.00	1.00	13
nazhan	1.00	1.00	1.00	17
thathsara	0.96	1.00	0.98	26
umesh	0.95	0.95	0.95	22
accuracy			0.99	141
macro avg	0.99	0.99	0.99	141
weighted avg	0.99	0.99	0.99	141

Model Accuracy

The face recognition model demonstrates excellent performance, achieving a training accuracy of 92% and a test accuracy of 98%.

3.3)Face Recognition Libraries

Face recognition libraries are software frameworks that provide pre-built functions and algorithms to detect, recognize, and analyze faces in images or videos. These libraries simplify the development of face recognition systems by offering efficient and optimized implementations of complex tasks like facial feature extraction, face alignment , and matching.

1.OpenCV : A popular open-source computer vision library that provides tools for face detection and recognition using Haar cascades and deep learning-based DNN modules.

2.dlib: Known for its powerful machine learning algorithms, dlib includes face detection and landmark estimation features that are crucial for face alignment and recognition.

3.Face Recognition Library (Python): Built on dlib, this library provides a simple interface for face recognition tasks like identifying and comparing faces.

4.TensorFlow: General-purpose deep learning frameworks that allow building custom face recognition systems using pre-trained models.

3.4)Attendance Marking in Excel

This project redefines attendance management by integrating advanced facial recognition technology with real-time automation, paired with seamless Excel-based record-keeping. By utilizing a pre-trained deep learning model, the system captures live video streams through a webcam, detects faces, and accurately identifies individuals in real-time.

Once a face is recognized, the system records attendance directly into an Excel file. Each entry includes comprehensive details such as the individual's name, ID, date, time, and punctuality status (e.g., "On Time" or "Late"). The structured Excel format ensures ease of access, organization, and compatibility for further analysis or reporting.

	A	B	C	D	E
1	Name	ID	Date	Time	Status
2	Himethma	ID003	########	15:53:08	Late
3	Dushara	ID006	########	15:53:15	Late
4	Kavindi	ID002	########	15:53:37	Late
5	Minhaj	ID001	########	15:53:46	Late
6	Umesh	ID008	########	15:53:47	Late
7	Nazhan	ID005	########	15:54:09	Late
8	Thathsara	ID004	########	15:54:20	Late
9					
10					

4)Results

The results of this project showcase the remarkable performance and real-world applicability of the face recognition and attendance marking system. By integrating advanced machine learning techniques with practical implementation, the system effectively achieves its primary goals. It not only ensures high accuracy in identifying individuals under various conditions but also seamlessly automates attendance recording with precise timestamps and status differentiation. The thoughtful incorporation of data augmentation, regularization, and optimization strategies has resulted in a robust model that balances efficiency and accuracy. These outcomes highlight the system's ability to improve attendance management, offering a reliable, easy-to-use solution that combines technology with practicality.

Face Recognition Accuracy

The system demonstrated high accuracy in identifying faces from both the known face dataset and real-time webcam input. The use of pre-trained face encoding and a deep learning model contributed to reliable recognition under various lighting conditions and angles. The confidence threshold (set to 80%) ensured only confident predictions were considered, minimizing misclassification errors.

Metrics Observed: Recognition accuracy was measured using the testing dataset and real-time scenarios, showing consistent performance across different faces.

Attendance Logging

The system successfully recorded attendance in an Excel-compatible CSV file, capturing key details such as name, ID, date, time, and attendance status ("On Time" or "Late"). The attendance marking mechanism effectively differentiated between recognized and unrecognized faces, ensuring accurate and reliable logs.

All recognized individuals were accurately logged, and the late threshold time (9:30 AM) was correctly applied to assign the attendance status.

Performance Evaluation

Training and Validation Accuracy: During the training phase, the model achieved satisfactory accuracy levels for both training and validation datasets. The use of learning rate scheduling, regularization, and data augmentation played a vital role in reducing overfitting and ensuring generalization.

Loss Curves: The training and validation loss curves converged steadily over the epochs, demonstrating stable learning. The final loss values were low, indicating effective optimization.

Real-Time Efficiency: The system processed live video input in real-time, ensuring a smooth and responsive interface for face detection and recognition.

Challenges and Limitations.

Due to the limitations of a small dataset, using pre-trained models or advanced architectures like the VarKFaceNet was not feasible. These models typically require extensive datasets to achieve optimal performance. As a result, I opted to design a simpler Convolutional Neural Network (CNN) architecture tailored to work effectively with the limited data available while still achieving reliable results.

Lighting and Angle Variations: Though the model was robust, extreme lighting conditions or severe face occlusions occasionally impacted recognition accuracy.

Processing Speed: On lower-end hardware, the system experienced minor lags during real-time detection, emphasizing the importance of optimized algorithms for deployment.

The project successfully met its objectives by developing an accurate, real-time face recognition system integrated with attendance logging. The results validate the effectiveness of combining face encoding techniques with a deep learning model for robust recognition and attendance management. Future improvements could include model optimization for faster processing and expanding the training dataset for better generalization.

5)Discussion

The face recognition-based attendance system was developed to streamline the process of tracking attendance, ensuring accuracy and efficiency while leveraging modern computer vision techniques. The project aimed to address key challenges associated with real-time facial recognition systems, including limited dataset size, data diversity, and system scalability. This discussion outlines the technical implementation, challenges faced, and the outcomes of the project.

Model Implementation

The model architecture for face recognition was designed with simplicity and efficiency in mind, using a Convolutional Neural Network (CNN) tailored to the project's dataset. Due to the limited dataset size, pre-trained models such as VGGFace or VarKFaceNet were not employed; instead, a CNN was developed to provide a balance between computational efficiency and recognition accuracy.

The final architecture included:

Data Augmentation Layer: Enhanced dataset diversity through techniques such as random zoom, rotation, and flipping, improving model generalization to unseen data.

Convolutional and Pooling Layers: Extracted essential features from facial images while reducing spatial dimensions, enabling the model to focus on high-level patterns.

Dropout and Regularization: Mitigated overfitting by introducing dropout layers and L2 regularization, particularly important for small datasets.

Dense Layers: Mapped extracted features to class probabilities using softmax activation, allowing for multi-class classification.

The system was trained and validated using an 85-15 split for the dataset, further partitioning the training set into training and validation subsets (70-15 split). To ensure efficient training, the data was normalized, and hyperparameters such as learning rate and batch size were optimized.

Key Challenges and Solutions

1**. Dataset Size and Variability**

One of the primary challenges was the limited size of the dataset, which restricted the ability to train deep models effectively. This limitation was mitigated using data augmentation techniques, which artificially increased dataset size by applying transformations to the existing images. This approach enhanced the diversity of the dataset and improved the model's robustness to variations in lighting, pose, and facial expressions.

3. **Computational Constraints**

Given the limited computational resources, the model was designed to balance performance and efficiency. Instead of using computationally intensive pre-trained models, the CNN achieved satisfactory accuracy with fewer parameters.

Evaluation Metrics

The model was evaluated using standard metrics such as accuracy, precision, recall, and confusion matrices. Key highlights of the results include:

Training and Validation Accuracy: The model achieved high accuracy on both the training and validation sets, indicating effective learning and generalization.
Testing Accuracy: Performance on the test set confirmed the model's robustness to unseen data.
Precision-Recall and ROC Curves: Demonstrated the model's ability to differentiate between classes effectively, even in challenging scenarios.

Additionally, confusion matrices revealed insights into the system's performance across various classes, highlighting areas for improvement and fine-tuning.

Integration with Attendance System

The face recognition component was seamlessly integrated into the attendance marking system. The workflow included:

Capturing real-time images or uploading pre-captured images.

Running predictions using the trained CNN model to identify individuals.

Marking attendance in a CSV file and updating the MongoDB database.

The integration ensured that the system could handle real-time data while maintaining scalability and reliability.

Strengths of the System

Efficiency: The custom CNN architecture provided a balance between computational efficiency and accuracy, making it suitable for deployment on limited hardware.

Scalability: The use of MongoDB for database storage allows the system to scale effortlessly to handle larger datasets in the future.

Real-Time Performance: The system demonstrated quick and reliable attendance marking, essential for practical applications in schools, workplaces, and events.

Future Work

While the system met its objectives, there are areas for future improvement:

Dataset Expansion: Incorporating larger and more diverse datasets will improve recognition accuracy and robustness.

Incorporating Pre-Trained Models: With access to more resources, models such as VGGFace or FaceNet can be fine-tuned for enhanced performance.

Real-World Testing: Deploying the system in live environments will provide valuable feedback for refining the model and integration workflows.

Ethical Considerations: Ensuring user consent and secure data handling will enhance the system's acceptance and usability.

This project successfully implemented a face recognition-based attendance system tailored to small datasets and computational constraints. By leveraging a custom CNN architecture, data augmentation, and robust training techniques, the system achieved reliable performance in marking attendance. Its seamless integration with CSV and MongoDB storage further underscores its potential for real-world applications. This project demonstrates the capability of deep learning techniques to innovate traditional processes, paving the way for more intelligent and automated systems in the future.

6)Conclusion

The face recognition-based attendance system developed in this project demonstrates the potential of leveraging modern deep learning techniques for automating and enhancing traditional processes. By implementing a custom Convolutional Neural Network (CNN), the system was tailored to the constraints of a small dataset and limited computational resources while achieving satisfactory performance in real-time attendance marking.

The project successfully addressed several challenges, including limited dataset size, overfitting, and variability in facial features. These issues were mitigated through the use of data augmentation, dropout, and regularization techniques, ensuring the model's robustness and generalization capability. The integration of the model with attendance recording mechanisms, such as CSV files and MongoDB databases, allowed for seamless and scalable operation.

The evaluation metrics, including accuracy, precision, recall, and confusion matrix analysis, confirmed the effectiveness of the system. The precision-recall and ROC curves further validated its reliability in distinguishing between different classes. The system's user-friendly interface, real-time capabilities, and scalability make it a viable solution for various applications, such as educational institutions, corporate environments, and event management.

In conclusion, this project illustrates the practical application of deep learning in solving real-world problems, specifically in the domain of facial recognition. The system not only provides an efficient and automated alternative to manual attendance marking but also lays the foundation for future advancements. Expanding the dataset, incorporating pre-trained models, and deploying the system in real-world scenarios will further enhance its utility and performance. This project highlights the transformative impact of artificial intelligence and sets the stage for continued exploration in the field of smart systems.

07)References

CNN Explainer: A highly interactive tool for visualizing CNNs, helping users understand how convolutional layers, filters, and activations work in real time. Visit CNN Explainer【15】.

Google Scholar: https://scholar.google.com.

VarKFaceNet Documentation: Highlights methods for loss functions such as Softmax and ArcFace, emphasizing their roles in improving intra-class variance and optimization during training, particularly for tasks like face recognition(VarKFaceNetAn_Efficient…).