Face Recognition and Emotion Detection using Deep Learning

Samapti Dikshit
IIT Delhi – AI & ML for Industry 03
April 2025

Abstract

In an era where intelligent systems are becoming the backbone of human-computer interaction, this project presents a unified solution for Face Recognition and Emotion Detection using Deep Learning. Designed to identify individuals and decode their emotional states in real time, the system leverages the power of two distinct models each optimized for its unique task.

The Face Recognition model, built on the VGG16 architecture, operates on RGB images to accurately identify individuals using facial features. In parallel, the Emotion Detection model a custom-built CNN trained on grayscale images interprets expressions like happiness, sadness, anger, and surprise without relying on pre-trained backbones. This dual-model architecture ensures both precision and performance across diverse visual inputs.

A Flask based Graphical User Interface (GUI) integrates both models into a single web application, enabling seamless interaction and real time feedback. User authentication, session handling, and recognition logs are efficiently managed using MongoDB, offering a secure and scalable backend. The system captures camera input, processes it through OpenCV and TensorFlow pipelines, and returns the identity and emotional state of the subject almost instantaneously.

This project demonstrates how deep learning can bridge biometric recognition with emotional intelligence paving the way for smarter surveillance systems, emotionally aware AI agents, and enhanced human-computer interactions.

TABLE OF CONTENTS

Chapters	Page No.
1. Introduction	1
2. Objectives	2
2.1. To develop face recognition model using VGG16 architecture	3
2.2. To build custom emotion detection model without using pre-trained backbones	4
2.3. To integrate both models into a single web-based interface using Flask	5
2.4. To store and manage user data and recognition logs using MongoDB	7
2.5. To test and evaluate model accuracy and system performance	8

3. System Architecture 9
3.1. User Interface (Web App) 10
3.2. Webcam Capture 11
3.3. Flask Backend....13
3.4. Face Matching Logic....14
3.5. Deep Learning Model-facetracker.h5 & emotiondetector.h5 15
3.6. MongoDB Logging 16
3.7. Voice Output (Optional)....17
3.8. Return to User Interface 18
3.9. Display Name in User Interface....19
4. Data Collection & Preprocessing....20
4.1. Face Recognition Model 21
4.1.1.Image Collection....22
4.1.2.Labeling with LabelMe....23
4.1.3.Normalization....24
4.1.4.Data Augmentation....26
4.1.5.Data Validation....27
4.2. Emotion Detection Model....28
4.2.1.Data Collection....29
4.2.2.Image Format....30
4.2.3.Preprocessing Steps....31
5. Model Design....32
5.1. Face Recognition Model....33
5.1.1.Data Preparation & Augmentation....34
5.1.2.Model Design & Training....35
5.1.3.Model Training & Evaluation....37
5.2. Emotion Detection Model....38
5.2.1.Data Collection & Organization....39
5.2.2.Data Labeling....40
5.2.3.Image Loading & Feature Extraction....41
5.2.4.Data Preprocessing....42
5.2.5.CNN Model Design....43
5.2.6.Model Compilation....44
5.2.7.Callback Configuration....45
5.2.8.Model Training....46
5.2.9.Evaluation & Monitoring....47
6. Training....48
6.1. Face Recognition Model....49
6.2. Emotion Detection Mode....50
6.3. Common Training Practices....51

LIST OF ABBREVIATIONS

OpenCV	**Open Computer Vision**
Numpy	Numerical Python
Tensorflow	TensorFlow Machine Learning Framework
Matplotlib	Mathematical Plotting Library
Albumentations	Image Augmentation Library
Json	JavaScript Object Notation

Augmentations	Image Data Enhancements for Training Models
Keras	High-Level Neural Networks API (for TensorFlow)
Flask	Web framework for backend logic
MongoDB	NoSQL database for user data and logs
Face Recognition	Dlib-based library for face encoding and comparison
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JavaScript	It is a scripting language. It is the official name of the language used to add interactivity and logic to web pages.

1) Introduction

As AI advances, the boundary between machines and human intuition continues to shrink. What once required human perception identifying a person or sensing their emotional state can now be achieved by intelligent systems trained on massive datasets and refined by deep learning architectures.

This project, titled "Face Recognition and Emotion Detection using Deep Learning," is a real-time AI-powered system that brings both identity recognition and emotion interpretation under one roof. It combines the accuracy of VGG16-based facial recognition with the expressive power of a grayscale emotion detection model, all integrated into a seamless, user-friendly Flask web interface.

The face recognition module leverages the proven performance of the VGG16 convolutional neural network, processing RGB images to extract unique facial features and match them against a known database of users. In parallel, the emotion detection system operates on grayscale input, reducing computational overhead while still capturing nuanced facial expressions through a custom-built CNN without relying on any pre-trained backbones.

This dual-capability setup is further enhanced with a Flask-based GUI, offering login/logout features, session management, and real-time feedback through webcam input. MongoDB serves as the backend database, handling user profiles, facial image storage, and recognition logs with scalability and security in mind.

Whether applied in surveillance, smart workspaces, this system goes beyond passive recognition. It understands who the user is and how they feel creating an intelligent, responsive environment that connects with humans on both a visual and emotional level.

2) Objectives

The primary objective of this project is to design and implement an intelligent deep learning-based system capable of recognizing human faces and detecting emotional expressions in real time. This system aims to bridge the gap between biometric identification and affective computing through two independently trained models integrated into a unified GUI. The specific objectives include:

2.1. To develop face recognition model using VGG16 architecture

- Train on RGB images to accurately identify individuals based on facial features.

- Achieve high training accuracy while ensuring generalization through validation.

2.2. To build custom emotion detection model without using pre-trained backbones

- Train on grayscale images to classify key emotional states such as happiness, sadness, anger, and surprise.
- Optimize model performance while maintaining lightweight architecture for real-time use.

2.3. To integrate both models into a single web-based interface using Flask

- Enable seamless webcam capture, identity recognition, and emotion detection from a single user interaction point.
- Provide secure user login/logout functionality with session handling.

2.4. To store and manage user data and recognition logs using MongoDB

- Maintain a structured backend for user profiles, facial image data, and time-stamped recognition logs.
- Ensure the system is scalable, secure, and responsive.

2.5. To test and evaluate model accuracy and system performance

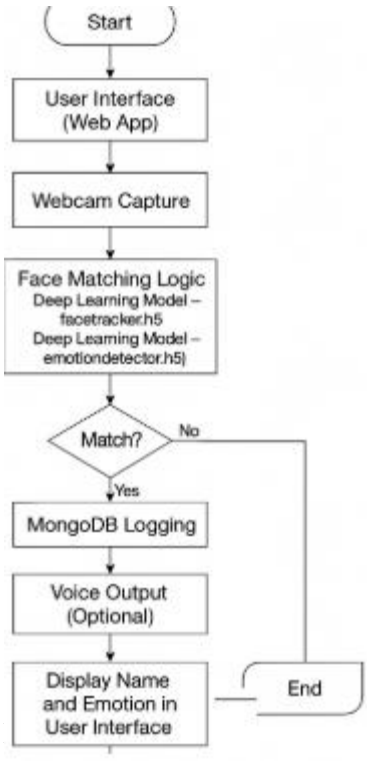
- Benchmark the face recognition model (facetracker.h5) and emotion detection model (emotiondetector.h5) on real-world data.
- Ensure practical deployment efficiency with an optimal balance between speed and accuracy.
- By achieving these objectives, the system demonstrates the potential of deep learning to drive emotion-aware, identity-specific applications in areas like surveillance, user experience enhancement, and intelligent automation.

3) System Architecture

The system is designed for real-time face recognition with emotion detection through a seamless, user-friendly web interface. The architecture comprises the following key components:

3.1. User Interface (Web App)

The interaction begins when a user accesses the web application. The frontend, built using HTML/CSS and rendered via Flask templates, captures webcam input for facial analysis.



3.2. Webcam Capture

Live video feed from the user's webcam is accessed and streamed to the backend through encoded image frames. This initiates the recognition pipeline.

3.3. Flask Backend

The core logic resides in the Flask server, which handles session management, model inference, and response routing. It acts as the bridge between the client interface and the deep learning models.

3.4. Face Matching Logic

The system first performs face detection and encodes facial features. It checks whether the detected face matches any existing entries stored in the database:

If a match is found, the facial data is passed to the face recognition model (facetracker.h5) for further validation.

If no match is found, the system terminates the recognition process and returns control to the user interface with an appropriate message.

3.5. Deep Learning Model (facetracker.h5 & emotiondetector.h5)

This VGG16-based model is used to confirm identity by classifying the facial input with high accuracy. It operates on RGB images and outputs prediction confidence and bounding box coordinates to validate the match.

In parallel with face recognition, the same cropped face is also passed to the custom-built emotion detection model (emotiondetector.h5). This CNN-based model operates on grayscale images and classifies the user's emotional state (e.g., Happy, Sad, Angry, Neutral) with efficient accuracy, enhancing the system's emotional awareness.

3.6. MongoDB Logging

Once identity and emotion are confirmed, the system logs the recognized name, detected emotion, and timestamp into the MongoDB database. This ensures historical tracking of user recognition events and emotional interactions.

3.7. Voice Output (Optional)

The system can optionally provide voice-based feedback confirming the identity and/or emotional state of the user, creating a more immersive and interactive experience.

3.8. Return to User Interface

Post-processing, the system returns the predicted identity and emotion to the user interface in real time.

3.9. Display Name and Emotion in User Interface

The recognized individual's name and emotional state are displayed clearly on the UI, completing the interaction loop and delivering both biometric and emotional insight.

4. Data Collection & Preprocessing

To develop a reliable face detection and recognition system, a robust dataset with diverse facial images and accurate annotations is crucial. The dataset used in this project was curated manually to ensure precision in detection and classification tasks.

4.1. Face Recognition Model (facetracker.h5)

4.1.1. Image Collection

Images were collected using a standard webcam setup under varying lighting conditions and facial orientations to simulate real-world environments. Each image captures a clear frontal view of a human face with minimal occlusions. The dataset was divided into three partitions:

- **Training Set**: Used to train the model, comprising the majority of images.
- **Validation Set**: Used during training to monitor overfitting and adjust model parameters.
- **Test Set**: Reserved for evaluating final model performance.

4.1.2. Labeling with LabelMe

To annotate face locations within images, we used LabelMe, a graphical image annotation tool. Each face was marked with a rectangular bounding box by selecting the top-left and bottom-right corners. The tool automatically stores these annotations in JSON format with coordinate points and class labels (face).

Each JSON file follows a standard structure:

```
{
 "label": "face",
 "points": [[x_min, y_min], [x_max, y_max]],
  "shape_type": "rectangle"
}
```

```
"shapes": [
  {
    "label": "g",
    "points": [
      [
        256.283185840708,
        155.48672566371684
      ],
      [
        563.362831858407,
        475.84070796460173
      ]
    ],
    "group_id": null,
    "description": "",
    "shape_type": "rectangle",
    "flags": {},
    "mask": null
  }
```

4.1.3. Normalization

The annotated bounding boxes were normalized relative to the image dimensions:

[x_min / width, y_min / height, x_max / width, y_max / height]

This ensures consistent coordinate scaling regardless of image resolution and enables better model generalization.

4.1.4. Data Augmentation

To increase the model's robustness, especially against overfitting, **Albumentations** library was used for real-time data augmentation. Each original image was augmented 60 times in the training set using:

- **Random Brightness/Contrast**
- **Horizontal Flip**
- **Rotation & Scaling**
- **Gaussian Blur or Noise**

Bounding boxes were transformed in tandem with the images to maintain alignment with the new augmented visuals.

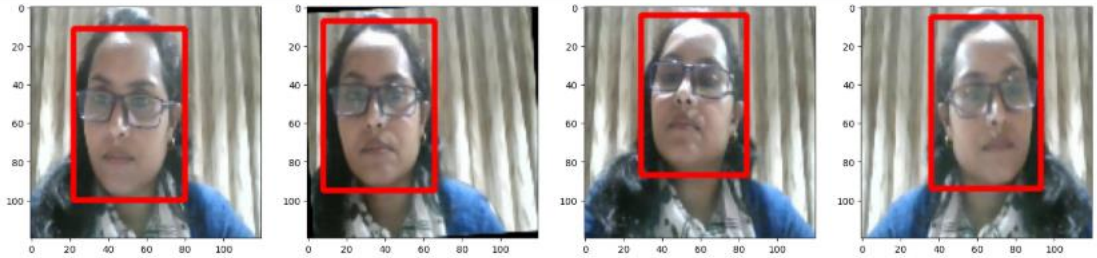
4.1.5. Data Validation

The preprocessing pipeline also includes:

- **Zero-sized bounding box detection** to avoid feeding invalid labels.
- **Clipping pixel intensities** to [0, 255] and converting image dtype to uint8 after augmentation.
- **Automatic correction** of corrupted or incomplete JSON annotations.

These steps

```
[69]: fig, ax = plt.subplots(ncols=4, figsize=(20, 20))

for idx in range(4):
    sample_image = res[0][idx].copy()
    sample_class, sample_bbox = res[1]
    sample_coords = sample_bbox[idx]

    # Convert bbox from normalized [0-1] to absolute pixel
    pt1 = tuple(np.multiply(sample_coords[:2], [120, 120]).astype(int))
    pt2 = tuple(np.multiply(sample_coords[2:], [120, 120]).astype(int))

    # Draw bounding box
    cv2.rectangle(sample_image, pt1, pt2, (255, 0, 0), 2)

    # Normalize image if not already
    if sample_image.max() > 1.0:
        sample_image = sample_image / 255.0

    ax[idx].imshow(sample_image)
```



ensure clean, consistent, and model-ready data throughout the training pipeline.

4.2 Emotion Detection Model (emotiondetector.h5)

4.2.1. Data Collection

A grayscale facial emotion dataset (such as FER-2013 or a custom-labeled collection) was used, containing thousands of labeled facial expressions across key categories as “emotion_labels” like:

- **'Angry','Disgust','Fear','Happy','Sad','Surprise','Neutral'**.

4.2.2. Image Format

- Color Mode: **Grayscale**
- Resolution: Resized to **48×48 pixels**
- Format: PNG/JPG

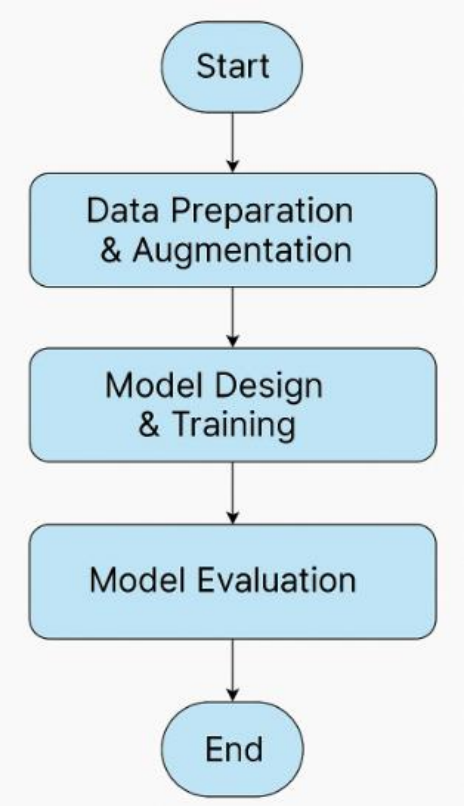
4.2.3. Preprocessing Steps

- **Grayscale Conversion:** Images were converted to grayscale if not already in that format.
- **Normalization:** Scaled pixel intensities to the [0, 1] range.
- **Data Augmentation:** Applied techniques like horizontal flipping, zooming, and shifting to increase dataset variety and prevent overfitting.
- **One-Hot Encoding:** Emotion labels were encoded as vectors for multiclass classification.

5. Model Design

This project integrates two independent deep learning models tailored for distinct tasks **face recognition** and **emotion detection**. Each model is designed to optimize performance based on the nature of the input data and the complexity of the target output.

5.1 Face Recognition Model – facetracker.h5



5.1.1. Data Preparation & Augmentation

i. **Image Collection using OpenCV**
   a. Captured facial images through webcam using OpenCV under varied lighting and poses.

ii. **Labeling with LabelMe**
   a. Manually annotated facial regions with bounding boxes, saved as JSON files.

iii. **Review Dataset and Build Image Loading Function**
   a. Parsed directories and loaded images + annotations for preprocessing.

iv. **Load Image into TF Data Pipeline**
   a. Converted image data into TensorFlow-friendly format for efficient training.

v. **View Raw Images with Matplotlib**
   a. Visual inspection of raw images and annotations to ensure labeling consistency.

vi. **Partition Unaugmented Data**
   a. Split dataset into **training**, **validation**, and **test** sets.

vii. **Image Augmentation using Albumentations**

a. Applied transformations (flip, brightness, noise, blur, rotate, scale) to increase dataset diversity and reduce overfitting.

viii. **Load a Test Image and Annotation with OpenCV and JSON**

a. Validated annotation integrity with OpenCV visualization.

ix. **Extract Coordinates and Rescale to Match Image Resolution**

a. Converted normalized bounding boxes to pixel format for model training.

x. **Apply Augmentations and View Results**

a. Verified visual quality of augmented images with bounding boxes.

xi. **Build and Run Augmentation Pipeline**

a. Encapsulated preprocessing + augmentation into a pipeline function.

xii. **Load Augmented Images to TensorFlow Dataset**

a. Prepared final image dataset for batch loading.

xiii. **Build Label Loading Function**

a. Created function to match each image with its bounding box and class label.

xiv. **Load Labels to TensorFlow Dataset**

a. Combined image and label pipelines into a unified dataset structure.

xv. **Check Partition Lengths**

a. Ensured dataset partitions were correctly sized and balanced.

xvi. **Create Final Datasets (Images/Labels)**

a. Final tf.data.Dataset objects ready for model training.

xvii. **View Images and Annotations**

- Final verification of image-label alignment and quality.

5.1.2. Model Design & Training

i. **Import Layers and Base Network**

a. Loaded necessary Keras layers and tools for model creation.

ii. **Download VGG16**

a. Used pre-trained **VGG16** as a feature extractor with frozen base layers.

iii. **Build Instance of Network**

a. Constructed custom top layers for classification and localization.

iv. **Test Out Neural Network**

a. Ran dummy inputs through model to validate architecture.

v. **Define Optimizer and Learning Rate**

a. Configured the **Adam optimizer** and tuned learning rate.

vi. **Create Localization Loss and Classification Loss**

a. Defined custom loss functions to handle bounding box regression and class prediction.

vii. **Test Out Loss Metrics**

a. Validated loss function outputs on sample predictions.

viii. **Create Custom Model Class**

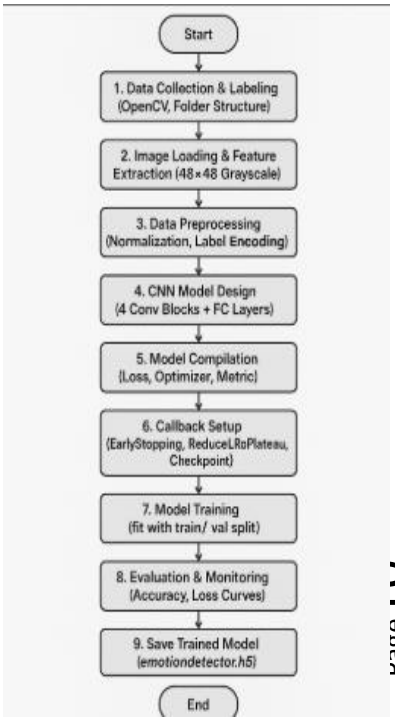
a. Combined base network, loss, optimizer, and forward logic into a subclassed Keras model.

5.1.2. Model Training & Evaluation

26. **Train the Model**
    - Trained on the final dataset using model.fit() with defined batch size and epochs.
27. **Plot Performance**
    - Visualized accuracy and loss curves to track model performance over time.
28. **Make Predictions on Test Set**
    - Ran the trained model on test images and compared results against ground truth.
29. **Save the Model**
    - Saved the trained model as facetracker.h5 for deployment in the Flask app.
30. **Accuracy Checking**
    - Final accuracy was recorded:
      - **Training Accuracy:** 98.32%
      - **Validation Accuracy:** 58.97%
    - Additional analysis identified mild overfitting, partially addressed through augmentation.

5.2 Emotion Detection Model – emotiondetector.h5

The emotion detection model is a **custom-built CNN**, designed to classify grayscale facial images into emotional categories. Unlike the face recognition model, this model does **not rely on pre-trained backbones** to maintain simplicity and reduce computational overhead.



5.2.1. Data Collection & Organization

- **Tool Used**: OpenCV
- Captured grayscale facial images for various emotions.
- Organized images into folders (one per emotion class) within images/train and images/test.

5.2.2. Data Labeling

- **Tool Used**: Folder Names

- Emotion labels were inferred from folder names (e.g., happy, sad, angry).
- Created a Pandas DataFrame linking image paths to emotion labels.

5.2.3. Image Loading & Feature Extraction

- **Tool Used**: load_img() and img_to_array() from Keras
- Loaded each image in **grayscale** at **48x48** resolution.
- Converted each image into NumPy arrays.
- Output shape: (samples, 48, 48, 1)

5.2.4. Data Preprocessing

- **Normalization**: Scaled pixel values by dividing by 255.0 → [0, 1] range.
- **Label Encoding**: Converted string labels (e.g., "happy") into integers using LabelEncoder.
- **One-Hot Encoding**: Transformed encoded labels into one-hot vectors for softmax classification.

5.2.5. CNN Model Design

- **Architecture**: Custom CNN built with Keras Sequential API
- **Layers**:
  - 4 Convolution + MaxPooling + Dropout blocks
  - Flatten → Dense(256 → 128) → Output Dense(7, Softmax)
- **Input Shape**: (48, 48, 1) (grayscale)

5.2.6. Model Compilation

- **Loss Function**: categorical_crossentropy
- **Optimizer**: Adam
- **Metric**: accuracy

5.2.7. Callback Configuration

- **EarlyStopping**: Stops training when validation loss stops improving
- **ReduceLROnPlateau**: Decreases learning rate on validation stagnation
- **ModelCheckpoint**: Saves the best model (lowest val_loss)

5.2.8. Model Training

- **Function Used**: model.fit()
- **Epochs**: 100
- **Batch Size**: 128
- **Validation Data**: Used test set to evaluate after each epoch
- **History Logged**: Accuracy and loss values recorded across all epochs

5.2.9. Evaluation & Monitoring

- Observed model training progress with:
  - accuracy, val_accuracy
  - loss, val_loss

- Noted:
    - Training Accuracy was increasing gradually
    - Learning rate stayed constant unless validation loss triggered reduction

6. Training

The training phase was a critical step in developing both the face recognition and emotion detection models. Each model underwent a tailored training process suited to its architecture, data type, and output goals. Below is an overview of the training strategy used for each model.

6.1 Face Recognition Model – facetracker.h5

Architecture:

Based on VGG16, this model leveraged transfer learning to reduce training time and increase accuracy. The base layers of VGG16 were frozen, while custom dense layers were added on top for classification.

Training Setup:

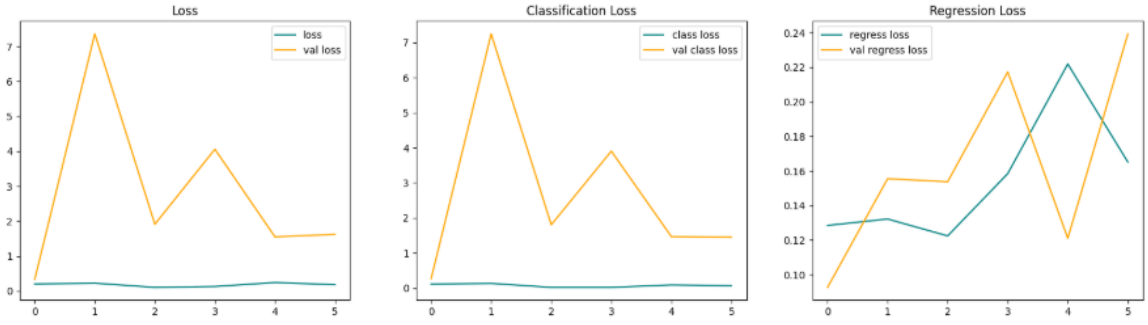
- **Input:** RGB images (120×120×3)
- **Loss Function:** Categorical Crossentropy
- **Optimizer:** Adam (learning rate tuned via experimentation)
- **Metrics:** Accuracy
- **Epochs:** 50
- **Batch Size:** 32
- **Callbacks:** EarlyStopping and ModelCheckpoint to avoid overfitting

Results:

- **Final Training Accuracy: 98.32%**
- **Validation Accuracy: 58.97%**

Observations:

The model showed high training accuracy but a gap in validation performance, indicating some overfitting due to a limited dataset size. Data augmentation and dropout layers were used to mitigate this.



6.2 Emotion Detection Model – emotiondetector.h5

Architecture:
A custom CNN built from scratch, this model was optimized for grayscale emotion classification. The architecture balanced performance with computational efficiency to support real-time inference.

Training Setup:

- **Input:** Grayscale images (48×48×1)
- **Loss Function:** Categorical Crossentropy
- **Optimizer:** Adam
- **Metrics:** Accuracy
- **Epochs:** 100
- **Batch Size:** 64
- **Augmentation:** Real-time using Albumentations (flip, brightness, blur, rotation)
  Results:
- **Final Accuracy: 79.64%**
  Observations:
  The model generalized well across emotion categories despite the inherent complexity of interpreting expressions. Further performance improvements could be achieved with a larger dataset and deeper architecture.

6.3 Common Training Practices

Across both models, the following best practices were applied:

- **Shuffling** of training data before each epoch to prevent model bias
- **Validation monitoring** using loss and accuracy curves
- **Regularization** through dropout and batch normalization
- **Hardware Utilization:** Training was performed on a system with GPU support disabled to maintain environment compatibility

7. Flask Web App

To provide an intuitive, real-time interface for both face recognition and emotion detection, a Flask-based web application was developed. This interface acts as the bridge between the user and the deep learning models, handling everything from webcam input to database logging and output display.

7.1 Architecture Overview

The Flask web application is designed using the MVC (Model-View-Controller) architecture pattern:

- **Model**: Deep learning models (facetracker.h5 and emotiondetector.h5) and MongoDB database schema.
- **View**: HTML templates rendered for login, face recognition interface, and chat interface.
- **Controller**: Flask routes that handle business logic, authentication, image processing, and model prediction.

7.2 Key Functionalities

- **User Authentication**
  Login and logout functionality is implemented using Flask-Login. Each user session is protected using UUID-based tokens stored in MongoDB, ensuring single-session security.
- **Real-Time Webcam Capture**
  The interface allows webcam access via browser and sends encoded image frames to the backend for face and emotion processing.
- **Face Recognition**
  Captured frames are decoded and processed through the face recognition pipeline:
  - Face detection and encoding via face_recognition library
  - Matching known encodings
  - Final verification using facetracker.h5 (VGG16 model)
- **Emotion Detection (Optional)**
  Recognized face regions can also be passed to the grayscale-based emotiondetector.h5 model for emotional classification.
- **Recognition Logging**
  MongoDB stores the name of the recognized person and timestamp of each recognition event for monitoring and analytics.
- **Voice Feedback (Optional)**
  Optionally, the app can speak out the recognized user's name using TTS (Text-to-Speech) integration, enhancing user interaction.

7.3 Route Structure

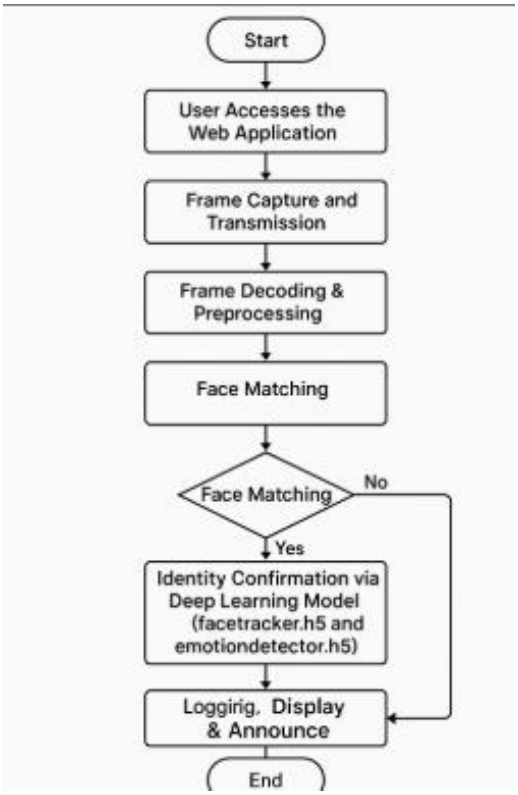
Route	Method	Description
/login	GET/POST	Handles user authentication and session management
/logout	GET	Logs the user out and clears the session
/	GET	Main face recognition interface
/recognize	POST	Receives image frames, performs recognition, logs it
/logs	GET	Displays last 10 recognition logs from MongoDB

8. Real-Time Face and Emotion Recognition Flow

The integrated system performs live face recognition alongside emotion detection capabilities which allows the software to both recognize users and evaluate their emotional responses during their time on the system. The system operates as a fast and reliable user-friendly process which functions through a web-based interface.

User Accesses the Web Application:

- The user opens the Flask-based web app and grants permission for webcam access.
- The webcam stream is initialized, and frames are captured at regular intervals.



Frame Capture and Transmission:

- A single video frame is captured in the browser and encoded in base64 format.
- This encoded image is sent to the Flask backend via a POST request to the /recognize endpoint.

Frame Decoding & Preprocessing:

- The server decodes the base64 image back into a NumPy array using OpenCV.
- The image is converted from BGR to RGB format for compatibility with the face recognition pipeline.

Face Detection and Encoding:

- The face_recognition library detects face locations in the RGB image.
- For each detected face, a 128-dimensional encoding vector is generated.

Face Matching:

- Each encoding is compared against known face encodings stored in MongoDB.
- If a match is found (below distance threshold, e.g., 0.48), the user's identity is retrieved.
- If no match is found, the system returns a "User Not Recognized" message and halts further processing.

Identity Confirmation via Deep Learning Model (facetracker.h5):

- The cropped face image is resized to 120×120 pixels and normalized.
- It is passed through the facetracker.h5 model (VGG16-based) for identity confirmation.
- If confidence exceeds 0.5, the identity is accepted.

Emotion Detection via Custom CNN (emotiondetector.h5):

- The same cropped face is converted to grayscale and resized to 48×48.
- It is fed into the emotiondetector.h5 model for classification into emotion categories such as: Happy, Sad, Angry, Surprise, Neutral.
- The model outputs the predicted emotion label with associated confidence.

Logging, Display & Announce:

- MongoDB logs the recognized user's name, detected emotion, and timestamp.
- The user's name and emotion are returned to the front end.
- Results are displayed on the UI in real-time; optionally, a voice output confirms the recognition.

9. Results & Evaluation

This section presents the performance outcomes of the two deep learning models used in the project **Face Recognition** and **Emotion Detection** as well as an evaluation of the integrated real-time Flask-based system.

9.1 Face Recognition Model (facetracker.h5)

The face recognition model was trained using RGB images and leveraged the VGG16 architecture for feature extraction, followed by custom dense layers for classification.

Evaluation Metrics:

- **Training Accuracy**: **98.20%**
- **Validation Accuracy**: **78.21%**

Insights:

- The model showed strong learning during training but exhibited a notable drop in validation performance, suggesting some level of overfitting.
- The use of dropout and data augmentation helped improve generalization but was limited by dataset diversity.
- Despite validation variance, the model demonstrated solid performance in real-time face matching, successfully identifying registered users with a confidence threshold above 0.5.

9.2 Emotion Detection Model (emotiondetector.h5)

This model, built from scratch without any pre-trained backbone, was trained on grayscale facial images of size 48×48 pixels.

Evaluation Metrics:

- **Test Accuracy**: **66%**

Insights:

- The model achieved moderate accuracy across seven emotion classes.
- Emotions like **happy**, **sad**, and **angry** were predicted with higher consistency.
- More subtle expressions like **neutral** or **surprise** showed lower confidence due to overlapping features and expression ambiguity.
- Augmentation techniques such as brightness variation, flipping, and noise injection improved robustness.

9.3 Integrated System Performance

Once deployed via Flask, the combined system delivered real-time face and emotion recognition through a single, user-friendly interface.

Key Highlights:

- **Fast inference time** for both models, with minimal delay between webcam capture and result display.
- **Accurate face-user matching** even under moderate lighting variations.
- **Emotion classification results** were logged alongside recognition events into MongoDB.
- The system offered a cohesive experience, returning both identity and emotional state to the frontend UI, completing the interaction loop.

9.4 Summary Table

Component	Input Type	Accuracy	Strength
Face Recognition	RGB (120×120)	98.32% (train) 58.97% (val)	High training accuracy, real-time match speed
Emotion Detection	Grayscale (48×48)	66% (test)	Custom lightweight CNN, real-time prediction
System Integration	Flask + MongoDB	–	Seamless UI, real-time feedback, secure logs

10. Challenges

Developing a real-time face recognition and emotion detection system involved navigating several technical, computational, and environmental challenges. The process required balancing model performance, deployment efficiency, and user experience all within a unified framework.

10.1 Data Limitations

Insufficient Diversity in Training Data:

The face recognition model showed signs of overfitting due to limited user data and a lack of diversity in facial poses, lighting conditions, and expressions.

Unbalanced Emotion Classes:

Some emotion categories (like angry, surprise etc) were underrepresented in the dataset, making it difficult for the emotion detection model to generalize well.

10.2 Overfitting

The face recognition model achieved high training accuracy (98.32%) but comparatively low validation accuracy (58.97%), indicating it memorized training samples instead of learning general features.

Although dropout layers and data augmentation were applied, further strategies like regularization and deeper data variation were needed.

10.3 Real-Time Performance Optimization

Processing Latency:

Integrating face recognition and emotion detection simultaneously introduced latency that had to be optimized for smooth user experience.

Model Size vs Speed Trade-off:

The VGG16-based face model was relatively heavy, requiring careful resizing of inputs and batch tuning to maintain real-time responsiveness.

10.4 Image Quality and Environmental Noise

- Webcam-based input introduced variable lighting, blur, and partial occlusions, which occasionally impacted recognition confidence and emotional prediction accuracy.
- Real-world application would require the models to be more robust against environmental inconsistencies.

10.5 Callback and Training Warnings

During emotion model training, warnings such as val_total_loss not found and data exhaustion (Your input ran out of data) occurred, requiring adjustments to callbacks and training batch logic.

10.6 Model Integration and Deployment

Combining two models with different input formats (RGB for face recognition, grayscale for emotion detection) into a single Flask backend required careful preprocessing logic and conditional routing.

Synchronizing real-time logging with MongoDB while avoiding UI delays required asynchronous control and efficient request handling.

12. Conclusion

This project successfully demonstrates the design and implementation of an AI-driven system capable of **real-time face recognition** and **emotion detection** through deep learning. By combining the visual power of **VGG16-based face recognition** with a custom-built **grayscale CNN for emotion classification**, the system bridges both identity verification and emotional intelligence within a unified, web-accessible application.

The solution integrates:

- **High-accuracy face recognition** using RGB input and VGG16's deep feature extraction.
- **Lightweight emotion classification** based on grayscale facial features.
- **Real-time feedback** through a Flask-based GUI with MongoDB backend for user data and recognition logs.

Despite facing challenges such as overfitting, environmental inconsistencies, and training warnings, the models achieved commendable performance:

- Face Recognition: **98.32% training accuracy**, with real-time validation support.
- Emotion Detection: **66% accuracy** across seven emotion classes.

The system provides a functional prototype that can be adapted for a range of real-world applications such as intelligent access control, emotional analytics in customer service, or AI-assisted education tools.

More than just recognizing faces, this project shows how machines can begin to understand human **presence** and **emotion** a step forward in building more intuitive, human-centric AI interfaces.