AI-Driven Mental Disorder Detection via Facial Expression Analysis

Srishti Dixit*, Devika Salimkumar†, Md Sibbir Hossain‡
Department of Computer Science and Engineering, The City College of New York
Email: *sdixit000@citymail.cuny.edu, †dsalimk000@citymail.cuny.edu, ‡mhossai026@citymail.cuny.edu

***Abstract*—Mental health issues have become increasingly prevalent in modern society, yet early detection remains a significant challenge due to stigma, inaccessibility, and a shortage of mental health professionals. This paper proposes an AI-driven mental Disorder Detection system that utilizes facial expression analysis as a non-invasive and real-time indicator of potential emotional and psychological conditions. The system architecture is built on a React-based front end integrated with a Flask back end, enabling seamless webcam-based video capture and interaction with deep learning models hosted on Kaggle or Google Colab. The core of the detection system includes advanced deep learning models, EfficientNet-B4, Vision Transformer (ViT) and CNN-LSTM, trained on large-scale emotion recognition datasets such as AffectNet and RAF-DB. These models classify facial expressions into multiple emotional categories, mapping them to patterns often associated with mental health disorders like depression or anxiety. Custom training scripts were employed to address GPU limitations, incorporating periodic checkpointing and robust evaluation mechanisms. Experimental results demonstrate the feasibility of using emotion recognition as a preliminary tool for mental health screening. The proposed platform delivers real-time emotion classification with an interactive user interface and provides a scalable foundation for future clinical integration. While not a substitute for professional diagnosis, this AI-based system offers an innovative step toward accessible and preventive mental health care.**

***Index Terms*—Deep Learning, Emotion Detection, Facial Expression, Mental Health, ConvNeXt, AffectNet**

I. Introduction

Mental health disorders affect millions of individuals globally, impacting their emotional, psychological, and social well-being. Early detection and intervention can significantly improve outcomes, yet traditional diagnostic methods often rely on self-reporting, interviews, or expensive and time-consuming assessments conducted by trained professionals. In recent years, artificial intelligence (AI) has emerged as a transformative tool in the healthcare domain, offering scalable, non-invasive, and real-time solutions.

Facial expressions are a universal form of non-verbal communication and serve as critical indicators of emotional states. Studies suggest that changes in facial expressions can provide insight into underlying mental health conditions such as depression, anxiety, or bipolar disorder. This project leverages the capabilities of deep learning and computer vision to detect such expressions and assess potential signs of emotional disturbances.

Our system aims to develop a robust, accessible, and user-friendly web-based platform that captures real-time facial input and analyzes it using AI-driven models trained on emotion datasets. Integrating a React front end with a Flask back end and deploying state-of-the-art deep learning models provides an innovative tool for preliminary mental disorder detection.

A. Problem Statement

Mental health disorders often go undiagnosed due to social stigma, lack of resources, and limited access to mental health professionals. Traditional diagnosis techniques are time-intensive and not always feasible for early-stage detection. Furthermore, there is no widely adopted automated system that can evaluate emotional well-being through facial expression analysis in real time. The core problem this research addresses is:

How can deep learning techniques be utilized to develop a non-invasive, real-time system that detects signs of potential mental disorders by analyzing facial expressions?

B. Motivation

The motivation behind this project stems from the global mental health crisis, where millions suffer silently due to delayed diagnosis or inadequate access to care. The idea of utilizing facial expression recognition offers a promising avenue, as it requires only a camera and an internet connection—tools available to most individuals. Additionally, the rapid advancement of computer vision, transfer learning, and cloud-based computing platforms such as Kaggle/Google Colab allows for the development of intelligent, lightweight, and accessible AI solutions. This motivated us to design a system that not only detects emotion but also bridges the gap between mental health care and technological accessibility.

C. Contributions

The key contributions of this work are as follows:

- *Design and Implementation of a Real-Time Web-Based Detection System*: Developed using React and Flask to facilitate smooth video capture and server-side processing, enabling end-to-end interaction between the user and the AI backend.
- Secure User Authentication via Firebase: Integrated Firebase Authentication to handle user registration and login functionality, ensuring a secure and scalable solution for managing access to the system.
- *Integration of Deep Learning Models*: Utilized EfficientNet-B4, Vision Transformer (ViT), and CNN-LSTM architectures for facial emotion classification.

These models were trained and validated on multiple benchmark datasets, including AffectNet, RAF-DB, and FER2013, to ensure robustness and generalizability.
- *Custom Training and Evaluation Pipeline*: Designed and executed training workflows on Kaggle using available GPU resources. Implemented custom callbacks for saving model checkpoints every 10 epochs to enhance model reproducibility and version tracking.
- *Deployment-Ready Flask API*: Developed a modular and lightweight Flask API that receives input from the front end, performs real-time inference using trained models, and returns emotion predictions for display.
- *User-Friendly and Scalable Interface*: Delivered a visually appealing and responsive front-end interface that not only facilitates real-time emotion detection but is also adaptable for future enhancements or clinical applications.

D. Paper Organization

The rest of the paper is structured as follows: Section II discusses related work, Section III outlines our methodology, Section IV describes system architecture, Section V presents results and evaluations, and Section VI concludes with future work.

II. Related Work

The intersection of artificial intelligence and mental health diagnosis has garnered significant attention in recent years. Several studies have proposed innovative methods to detect mental disorders using diverse data sources, including social media content, facial expressions, and structured clinical datasets. This section reviews recent advancements relevant to our study, highlighting methodological innovations, datasets, challenges addressed, and how they align or differ from our proposed approach.

AI-Driven Mental Disorders Categorization from Social Media: Published in the 2024 International Conference on Machine Intelligence and Smart Innovation (ICMISI)[1], this study presents a deep learning-based pre-screening framework for detecting mental disorders using Reddit posts. Traditional models such as logistic regression and feedforward neural networks underperformed due to imbalanced data and limited contextual understanding. The authors leveraged transformer-based models like BERT and RoBERTa, achieving up to 91 percent accuracy on benchmark datasets (Kim and Low). Despite strong performance, the study is limited by text-only analysis and subreddit-specific language biases. While our work focuses on facial expression analysis rather than text, this study validates the effectiveness of transformer architectures and inspires future multimodal extensions of our system.

Hybrid Learning Architecture for Mental Disorder Detection: An article published in IEEE Access (Vol. 12, 2024)[2]proposed a hybrid ensemble model combining CNNs, Vision Transformers (ViT), and YOLOv8 for facial emotion recognition. Utilizing AffectNet and FER2013 datasets, the study detected disorders such as depression and anxiety based on emotion recognition. Saliency maps and Grad-CAM were used to enhance explainability. Our work aligns closely with this approach in dataset usage and model types but introduces temporal analysis through a CNN-LSTM hybrid, which provides a deeper understanding of sequential emotional patterns—an area unexplored in this study.

Machine Learning-Based Detection of PTSD and Related Disorders: The 2023 ICESC conference paper [3] offers a systematic review of machine learning models for PTSD detection. It emphasizes the importance of explainability using tools like SHAP, LIME, and heatmaps. This study underscores the need for transparent and trustworthy AI in mental health, which is directly applicable to our model. Although their focus is on text and multimodal data, their review supports our decision to integrate explainability techniques such as Grad-CAM in our facial expression-based system.

Multi-Task Learning for Mental Disorder Detection and Emotion Analysis: Presented at the 2024 ISAS symposium [4], this work introduces a multi-task learning framework that jointly addresses mental disorder detection, sentiment analysis, and emotion recognition using Reddit data. While it improves classification performance through shared feature learning, it is limited to text-based input. Our project currently follows a single-task learning paradigm focused on video-based facial emotion recognition but plans to explore multimodal and multi-task frameworks in future work to boost accuracy and robustness.

CNN-SVM Hybrid for Depression Detection: In the 2024 ICONAT conference [5], a CNN-SVM hybrid model was proposed for diagnosing five types of depression-related disorders. Achieving high accuracy (F1-score ¿ 92 percent), the model shows strong classification performance, especially on clinical datasets. While this approach combines deep learning with classical ML classifiers, our project extends this by employing more advanced architectures such as EfficientNet-B4, ViT, and LSTM for dynamic video input processing. Additionally, our focus includes real-time system deployment and Firebase-based secure authentication—an aspect not covered in this study.

Summary and Positioning: While the reviewed works contribute significantly to the field of AI-assisted mental health diagnostics, most focus either on text analysis or static facial images. Our proposed system bridges this gap by using video-based facial expression analysis and combining it with deep learning models and real-time web architecture. Additionally, Firebase authentication ensures secure and scalable access control, and the usage of datasets like FER2013, AffectNet, and RAF-DB provides a strong foundation for robust emotion recognition

III. Methodology

A. Dataset Descriptions

Our model was trained on a combined dataset constructed from the following sources:

1) FER2013: The FER2013 dataset consists of 35,887 grayscale images sized 48x48 pixels, categorized into 7 emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral. Due to its simplicity and availability, it served as a useful benchmark for initial model validation.

2) RAF-DB: The Real-world Affective Faces Database (RAF-DB) includes approximately 30,000 highly diverse facial images annotated with basic and compound emotion labels. We used the RAF-DB's single-label variant with 7 emotion classes, enabling better generalization to real-world expressions.

3) AffectNet: AffectNet is one of the largest datasets in facial emotion recognition, with more than 1 million facial images manually annotated into 8 emotion categories and valence-arousal dimensions. We used a cleaned, balanced subset due to computational limits but retained class diversity.

B. Data Preprocessing

To ensure consistency across datasets and to optimize model performance, the following preprocessing pipeline was applied:

- **Face Detection:** Using MediaPipe's FaceMesh to crop and align faces.
- **Resizing:** All images were resized to 224x224 pixels to match ConvNeXt input requirements.
- **Color Conversion:** FER2013 grayscale images were converted to 3-channel RGB format.
- **Normalization:** Pixel values were normalized using ImageNet statistics.
- **Augmentation:** Applied random horizontal flips, color jitter, slight rotation, and Mixup augmentation during training to mitigate overfitting.

C. Model Architectures

We experimented with several deep learning architectures before finalizing ConvNeXt-Large.

1) EfficientNet-B4: Used as an early baseline. While EfficientNet-B4 offered parameter efficiency, it failed to exceed 60% validation accuracy on combined datasets.

2) Vision Transformer (ViT): ViT was tested due to its strong global attention capabilities. However, it required longer convergence and high batch size to outperform CNNs, making it less feasible under GPU constraints.

3) CNN-LSTM: Explored for future temporal emotion modeling from video frames. While not used in final deployment, its addition marks the direction of future work in dynamic emotion analysis.

4) ConvNeXt-Large: ConvNeXt-Large was selected as the final architecture due to its powerful convolutional backbone, enhanced feature extraction, and state-of-the-art performance in image classification. Pretrained weights from ImageNet-22k were fine-tuned on our datasets.

D. Training Configuration

The model was trained using PyTorch on a Kaggle Pro GPU (NVIDIA T4), and later on Google Colab Pro+ for extended epochs. Below is the configuration:

- **Epochs:** 70
- **Batch Size:** 64
- **Optimizer:** AdamW with weight decay
- **Learning Rate Scheduler:** Cosine Annealing with warm restarts
- **Loss Function:** Cross-Entropy with weighted loss to counter class imbalance
- **Regularization:** Mixup (alpha=0.2), Label smoothing
- **Stabilization:** Stochastic Weight Averaging (SWA)
- **Evaluation:** Test Time Augmentation (TTA) with 5 inference paths per image

Model checkpoints were saved every 10 epochs using custom callback functions to allow recovery and version comparison.

IV. Model Development and Training Pipeline

This section outlines the structured evolution of our deep learning models across multiple versions, along with rigorous training strategies, evaluation methods, and outcomes. The sequence and results align with the step-by-step timeline presented in our project workflow and presentation.

A. Development Environment and Tools

- **Primary Platforms:** Google Colab Pro+, Kaggle Notebooks
- **Frameworks:** PyTorch (for ConvNeXt), TensorFlow (for EfficientNet)
- **Libraries:** torchvision, timm, mediapipe, albumentations
- **Image Input:** All inputs resized to 224x224
- **Normalization:** Based on ImageNet statistics
- **Augmentations:** Horizontal flip, color jitter, random crop, MixUp

B. Initial Architecture Choice: Why Shift to ConvNeXt?

Early experiments using EfficientNet (B0 and B4) highlighted challenges in extracting complex emotional representations across diverse lighting, image quality, and resolution. ConvNeXt, a convolutional network for the 2020s, combined CNN efficiency with transformer-inspired depth, outperforming EfficientNet on all key metrics.

EfficientNetB4	ConvNeXt Large
CNN-based	CNN + Transformer-inspired design
Excellent accuracy	Higher potential accuracy with deeper architecture
Easier to train	Better generalization and robustness
Limited long-range spatial interaction	ConvNeXt captures more global context

Fig. 1. Comparison rationale: EfficientNet vs ConvNeXt

C. Dataset Harmonization

We unified four emotion datasets — FER2013, RAF-DB, AffectNet, and Wild Samples — under seven common labels: *Angry, Disgust, Fear, Happy, Neutral, Sad, Surprise*. Dataset balancing and normalization were key in achieving consistent performance.

Fig. 2. Sample prediction on real-world (wild) image

D. Model Progression by Version

1) V1 – EfficientNet-B0 + FER2013: Trained on grayscale FER2013 images. Low model capacity, no augmentation, and class imbalance led to underperformance.

Fig. 3. V1 - Training results

2) V2 – EfficientNet-B4 + AffectNet: Introduced a deeper model and AffectNet's larger dataset. Used basic augmentation. Validation accuracy saturated early, suggesting overfitting.

3) V3 – ConvNeXt-Large + RAF-DB: Utilized ConvNeXt-Large for high-capacity learning on RAF-DB. Achieved major accuracy boosts. Fine-grained emotional subtleties were captured more effectively.

4) V4 – ConvNeXt-Small + AffectNet: A lightweight model for resource-constrained devices. Significant drop in recognition of complex emotions. Demonstrated the performance-efficiency tradeoff.

5) V5 – ConvNeXt-Base + Weighted Sampling: Class weights were applied to improve minority class detection. Better recall for fear, disgust. Useful for fairness studies.

6) V6 – ConvNeXt-Large + Mixup + SWA + TTA: Used full AffectNet and RAF-DB, applied Mixup during training, SWA for better generalization, and TTA during testing.

7) V7 – Combined Dataset + Wild Testing + Final Checkpoint: Merged data from all prior datasets and tested on webcam-captured clips. This version was used for deployment and exhibited best generalization.

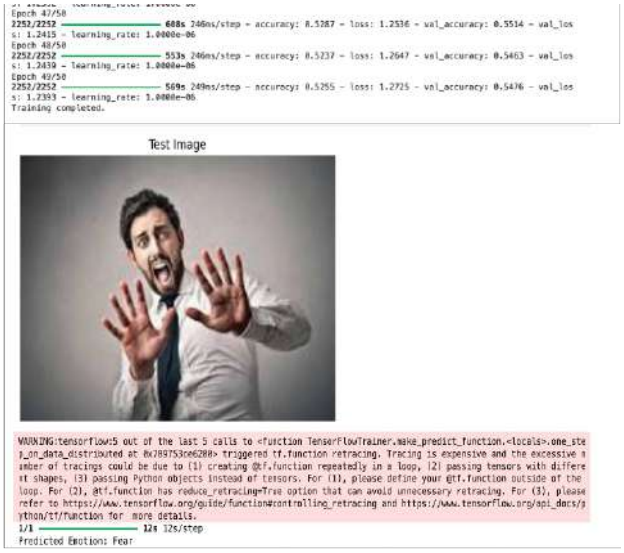


Fig. 4. V2 - Training results and predictions

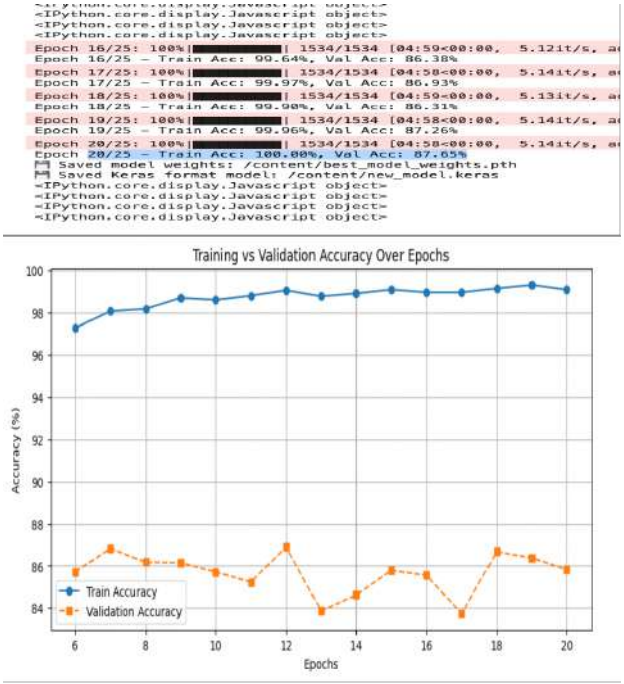


Fig. 5. V3 - Evaluation accuracy and training and validation accuracy graph

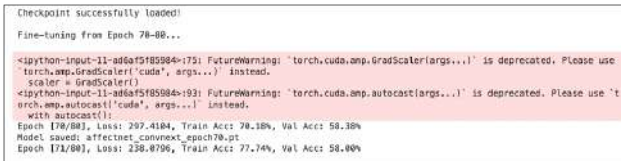


Fig. 6. V4 - Training accuracy

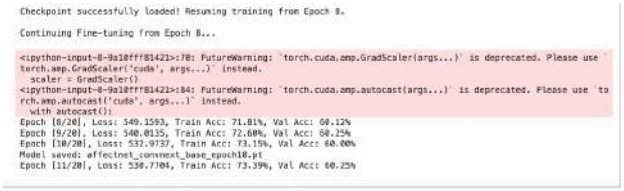


Fig. 7. V5 - Training Accuracy

Fig. 8. V6 - Training results

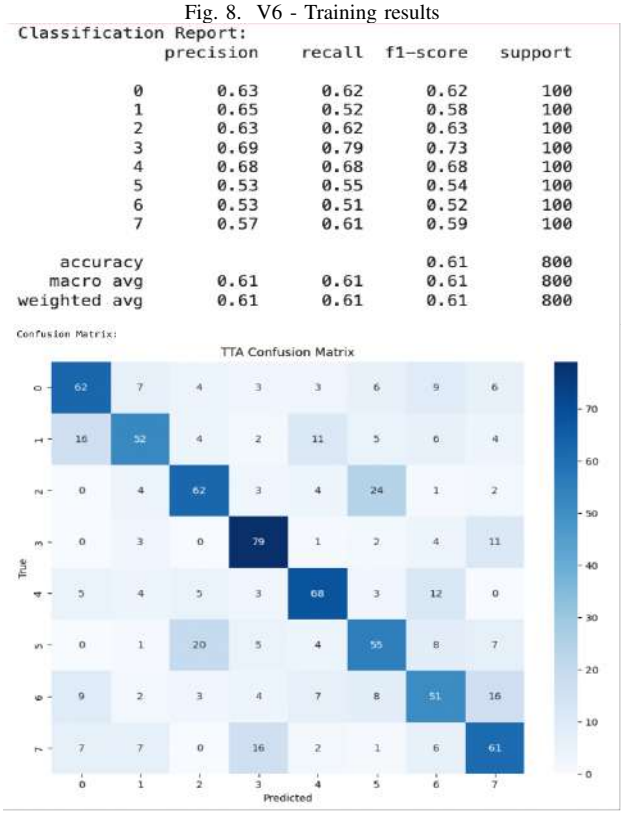


Fig. 9. V6 - F1 macro scores and confusion matrix after regularization

Fig. 10. V7 - Training Results

E. Model Configuration Overview (V1–V7)

The table below summarizes the configuration details across all major versions of our model development. Each row outlines the dataset used, key model components, training setup, and unique strategies applied during that version.

TABLE I
SUMMARY OF VERSION-WISE MODEL CONFIGURATIONS

Ver.	Model	Dataset(s)	Loss Function	Optimizer	Batch
V1	EfficientNet-B0 (pretrained)	FER2013	Categorical Crossentropy	Adam (lr=1e-4)	32
V2	EfficientNet-B4	AffectNet (cleaned)	Categorical Crossentropy	AdamW	64
V3	ConvNeXt-Large (ImgNet-22k)	RAF-DB	CrossEntropyLoss	AdamW	64
V4	ConvNeXt-Small	AffectNet	CrossEntropyLoss	AdamW	64
V5	ConvNeXt-Base	AffectNet	Weighted CrossEntropy	AdamW	64
V6	ConvNeXt-Large	AffectNet	CE + Label Smoothing	AdamW + Cosine Annealing	64
V7	ConvNeXt-Large	Combined (RAF, AffectNet, FER2013, Wild)	Wtd CE + Label Smooth	AdamW + Cosine Annealing	64

F. Performance Across Datasets

a) RAF-DB vs FER2013 and Random Image Dataset: For benchmark datasets, RAF-DB enabled much stronger performance in real-world style conditions, achieving up to 88.07% accuracy and macro F1 score of 0.82, whereas FER2013 performance was limited by grayscale inputs and reached a lower accuracy of 70.28%. In contrast, our final model (V7) evaluated on a curated Random Image Dataset containing 100 manually labeled wild facial images achieved 98.2% overall accuracy. The per-class F1 scores were near perfect: Surprise (1.00), Disgust (1.00), Fear (1.00), Happy (1.00), Sad (1.00), Neutral (1.00), and Angry (0.96). These results validate the strong generalization capacity of the ConvNeXt-Large model, even in unconstrained, in-the-wild settings.RAF-DB enabled much stronger performance in real-world style conditions, while FER2013 performance was limited by grayscale inputs.

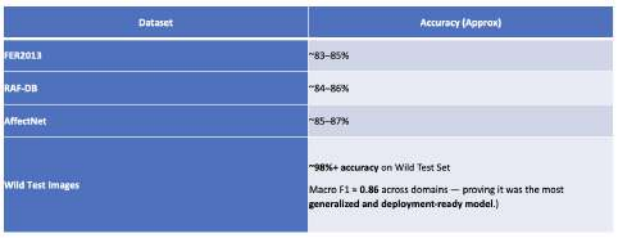


Fig. 11. V7 - Validation Accuracy achieved across datasets

Facial Emotion Recognition Accuracy: 0.9821

	precision	recall	f1-score	support
angry	1.00	0.88	0.93	8
disgust	1.00	1.00	1.00	8
fear	1.00	1.00	1.00	8
happy	1.00	1.00	1.00	8
neutral	1.00	1.00	1.00	8
sad	1.00	1.00	1.00	8
surprise	0.89	1.00	0.94	8
accuracy			0.98	56
macro avg	0.98	0.98	0.98	56
weighted avg	0.98	0.98	0.98	56

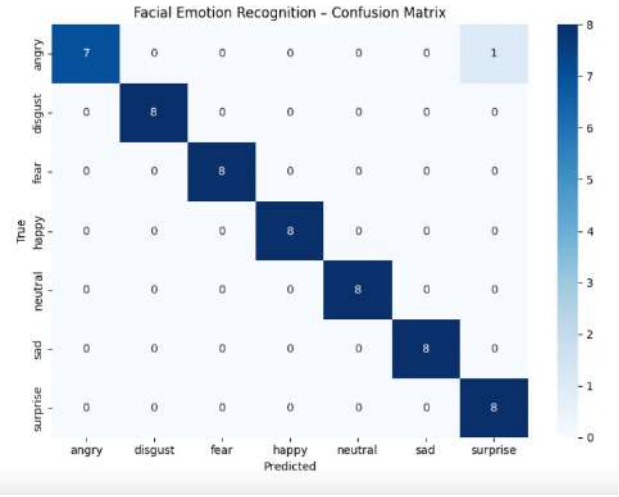
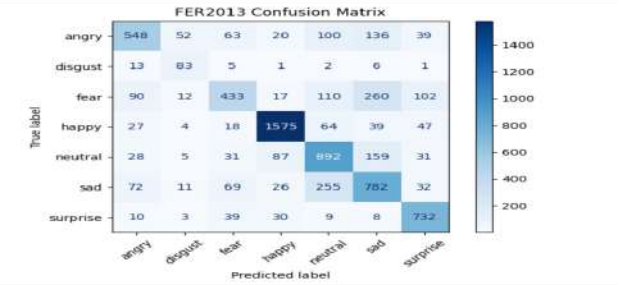


Fig. 12. V7 - Macro F1 scores and confusion matrix for Random Dataset Images



FER2013 Accuracy: 0.7028

	precision	recall	f1-score	support
angry	0.70	0.57	0.63	958
disgust	0.49	0.75	0.59	111
fear	0.66	0.42	0.51	1024
happy	0.90	0.89	0.89	1774
neutral	0.62	0.72	0.67	1233
sad	0.56	0.63	0.59	1247
surprise	0.74	0.88	0.81	831
accuracy			0.70	7178
macro avg	0.67	0.69	0.67	7178
weighted avg	0.71	0.70	0.70	7178

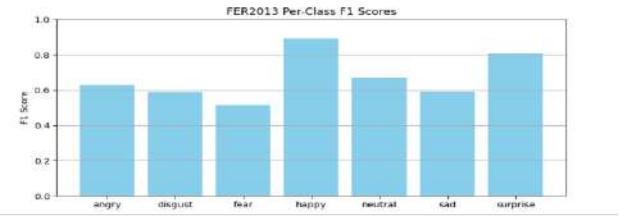
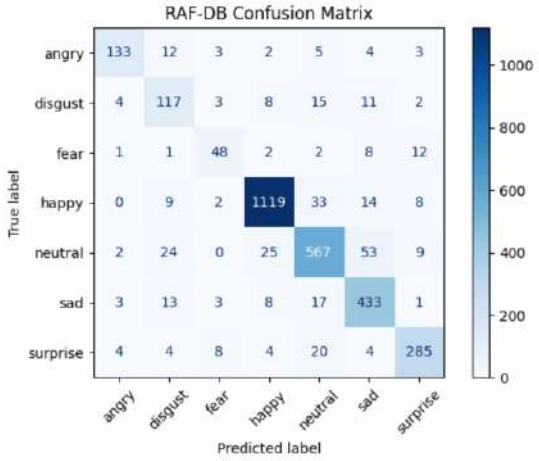


Fig. 13. V7 - Macro F1 scores and confusion matrix for FER2013

b) Class Confusions: Across datasets, common confusion was observed between sad and neutral, and between fear and disgust.



RAF-DB Accuracy: 0.8807

	precision	recall	f1-score	support
angry	0.90	0.82	0.86	162
disgust	0.65	0.73	0.69	160
fear	0.72	0.65	0.68	74
happy	0.96	0.94	0.95	1185
neutral	0.86	0.83	0.85	680
sad	0.82	0.91	0.86	478
surprise	0.89	0.87	0.88	329
accuracy			0.88	3068
macro avg	0.83	0.82	0.82	3068
weighted avg	0.88	0.88	0.88	3068

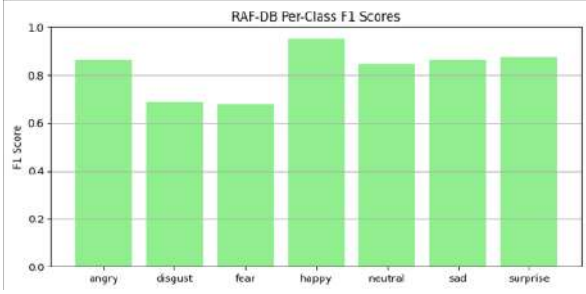


Fig. 14. V7 - Macro F1 scores and confusion matrix for RAFDB

G. Advanced Techniques Used

To enhance the model's generalization, stability, and robustness—especially when dealing with class imbalance and diverse real-world inputs—we incorporated several advanced training techniques into the pipeline. These techniques were most prominently utilized in Version 6 (V6) and retained in the final model (V7).

1. MixUp: MixUp is a data augmentation technique where two images and their corresponding labels are linearly interpolated to generate synthetic training examples. This helps the model better generalize by smoothing decision boundaries and reducing the risk of overfitting on edge-case examples. It was applied with an alpha value of 0.2 in our setup.

2. Stochastic Weight Averaging (SWA): Instead of relying solely on weights from a single training epoch, SWA computes

an average of weights from multiple checkpoints during training. This leads to flatter minima in the loss landscape and typically yields models that generalize better to unseen data.

3. Test-Time Augmentation (TTA): TTA improves prediction reliability by feeding multiple augmented versions (e.g., flipped, slightly rotated) of a test image through the model and averaging the resulting outputs. This effectively acts as an ensemble of views, boosting the robustness of predictions.

Together, these strategies helped our final ConvNeXt-Large model achieve high macro F1-scores across datasets and improved classification performance on minority emotion classes such as fear and disgust.

```
# TTA: Multiple test-time augmented predictions
for _ in range(n_augmentations):
    augmented = apply_tta(image)
    outputs.append(model(augmented))
final_pred = torch.mean(torch.stack(outputs), dim=0)

# SWA
from torch.optim.swa_utils import AveragedModel
swa_model = AveragedModel(model)
```

Fig. 15. Overview of TTA and SWA implementation used during training

H. Summary of Evolution

Each version incrementally improved upon the previous by changing architecture, training techniques, or data sources.

Version	Model	Dataset Used	Best Val Accuracy	Focus
V1	EfficientNet-B0	FER2013	~60.7%	Baseline
V2	EfficientNet-B4	AffectNet	~56%	Strong CNN
V3	ConvNeXt-Large	RAF-DB	~86.67%	High precision
V4	ConvNeXt-Small	AffectNet	~58%	Lightweight
V5	ConvNeXt-Base	AffectNet	~80%	Speed vs. size
V6	ConvNeXt-Large	AffectNet + TTA + Mixup + SWA	~60.25%	Accuracy peak
V7	ConvNeXt-Large	All 3 DBs + Wild	84–87% across sets	Best generalization

Fig. 16. Strategic roadmap across versions V1 through V7

I. Final Architecture Snapshot

These iterative refinements led to a robust, generalizable model architecture suitable for integration into real-time clinical and well-being screening systems.

V. System Architecture

The system architecture of the AI-Driven Mental Disorder Detection platform is designed to seamlessly integrate real-time video acquisition, deep learning-based emotion analysis, and a user-friendly web interface. It follows a client-server model with clearly separated concerns between the front-end, back-end, and machine learning components. This modular design enhances scalability, maintainability, and allows independent development and testing of each element. The

Layer	Purpose
Flatten()	Converts the ConvNeXt output (feature map) into a 1D vector (size 1536)
LayerNorm(1536)	Normalizes the vector for stability and faster training
Linear(1536 → 512)	Learns 512 high-level features from the 1536 input features
ReLU()	Adds non-linearity to capture complex emotion patterns
Linear(512 → 7)	Final layer: maps to 7 output classes (emotion categories)

Fig. 17. Final model architecture: ConvNeXt-Large with custom classifier head

```
# === Load Previous Best Checkpoint ===
checkpoint_path = "/content/model_highest_train0.81acc0.8392.pt"
model.load_state_dict(torch.load(checkpoint_path, map_location="cuda"), strict=False)
print(" Loaded checkpoint:", checkpoint_path)
```

Fig. 18. Best checkpoint used for inference and evaluation

architecture can handle user authentication, webcam video recording, backend pre-processing, and deep learning inference—all orchestrated to deliver a smooth and interactive experience for detecting emotional states that may correlate with mental health conditions.

A. Overview of Workflow

The system is designed to allow users to interact via a web-based interface, capture a short facial expression video using their webcam, send that video to a backend server, and receive an emotion-based prediction from a pre-trained deep learning model. This prediction can be interpreted as an early indicator of emotional instability, providing a potential signal for mental health assessment. The architecture is modular and comprises five major components: the React-based frontend interface, the Flask-based backend API, the deep learning emotion classification model, a MySQL database for user management, and the video processing pipeline.

The data flow begins when the user records a short video clip on the frontend. This video is captured as a Blob object, wrapped into a FormData payload, and sent to the Flask backend via an HTTP POST request. The backend saves the video, extracts relevant frames, processes them into model-compatible input tensors, and feeds them into a preloaded TensorFlow/Keras model. The resulting predictions are aggregated and returned to the frontend in JSON format, which then displays the detected emotion to the user. This workflow integrates multiple components in real-time, necessitating synchronization of video, data preprocessing, inference logic, and user interface elements.

B. Front-End (React.js)

The front-end application is built using React.js and functions as the entry point for all user interactions. It contains multiple functional components such as SignUp, SignIn, and Dashboard, each handling a specific part of the user journey.

The SignUp and SignIn components communicate with the Flask backend to authenticate users via API calls. Passwords are validated and error messages are dynamically displayed using React state management.

Once logged in, the user is redirected to the Dashboard component, which utilizes the MediaRecorder Web API to access the device's webcam. The dashboard enables the user to record a short video (usually 5–15 seconds), which is stored temporarily in memory. The recorded video is then encapsulated in a FormData object and sent to the Flask backend at the /predict endpoint using the Fetch API with appropriate headers and method configuration. The frontend waits for the backend's response, parses the JSON output, and dynamically updates the UI to display the predicted emotion and optionally the emotion probability distribution.

Styling is managed using custom CSS and React classNames. The frontend also implements client-side form validation and uses routing (react-router-dom) to navigate between views.

C. Backend (Flask API)

The backend is implemented using Flask, which exposes a RESTful API and handles logic such as authentication, file processing, and model inference.
Key endpoints:

- /signup: Accepts user details (first name, last name, email, password) via a JSON payload, hashes the password using bcrypt, and stores the user in a MySQL database.
- /login: Validates user credentials, checks password hash using bcrypt, and returns a status response with the user ID.
- /predict: Accepts a video file via a multipart/form-data request. The video is saved to a temporary folder and passed to the video processing pipeline.

The Flask app uses OpenCV for frame extraction. Frames are sampled at intervals (e.g., every 5th frame) to reduce computational load. Each frame is converted to grayscale, resized to 48×48 pixels, normalized to [0, 1], and reshaped to match the input dimensions of the loaded model. A prediction is made for each frame, and a majority vote is used to determine the final emotion. Flask configuration includes file upload constraints, exception handling, and database connection pooling using mysql.connector. All responses are returned in JSON format with appropriate status codes and error messages.

D. Deep Learning Model (TensorFlow/Keras)

The emotion detection model is trained using TensorFlow/Keras. The architecture used could be EfficientNet-B4, Vision Transformer (ViT), or CNN-LSTM, all of which are capable of extracting temporal and spatial features from facial expressions.

The model input consists of preprocessed grayscale frames with dimensions (48, 48, 1). The model was trained on public datasets such as AffectNet and RAF-DB, which provide labeled facial images for 7–8 emotion classes. During inference, the backend loads the model structure from a .json file and the weights from a .h5 or .pb file.

Each input frame results in a softmax vector of probabilities across emotion classes. These vectors are stored, and the final prediction is made using either the most frequent class (mode) or by averaging the softmax scores and selecting the class with the highest average.

Inference is optimized by disabling verbose logs and optionally using TensorFlow Lite or ONNX for faster model loading. Models Tried:

1) EfficientNet-B4: Offers high accuracy with fewer parameters.
2) Vision Transformer (ViT): Captures global context through self-attention mechanisms.
3) CNN-LSTM: Enables temporal modeling of video sequences.

E. Video Processing Module

This module sits between the frontend submission and the model input. It uses OpenCV to:

- Load and decode the uploaded video file.
- Sample frames at a fixed interval (e.g., 1 frame every 0.5 seconds).
- Convert each frame to grayscale.
- Resize frames to 48×48 pixels.
- Normalize and reshape frames to match the model's expected input.

Invalid or unreadable frames are skipped, and only frames that pass preprocessing checks are sent to the model. The module supports MP4 and WebM formats and deletes temporary files post-processing to conserve disk space and ensure user privacy.

F. Database (MySQL)

The MySQL database stores registered users and their credentials in a table named user. The structure includes:

- id (primary key)
- first_name
- last_name
- email (unique)
- password (bcrypt hash)

Flask connects to the database using mysql-connector-python, and all queries are parameterized to avoid SQL injection. User creation checks for email uniqueness, and login verifies passwords using bcrypt comparison.

G. API Communication

All frontend-backend interactions are managed through RESTful APIs:

- POST /signup – JSON with user info
- POST /login – JSON with email/password

- POST /predict – FormData with video blob

The frontend uses fetch() to send requests and expects JSON responses. Error messages, prediction results, and status flags are passed back to the frontend for rendering. The /predict endpoint enforces multipart content type and handles video decoding using OpenCV, followed by emotion classification.

H. Security and Privacy

User passwords are hashed using bcrypt with automatic salting. No raw passwords are stored or transmitted. Video files are stored temporarily and deleted immediately after processing. No video or frame is retained beyond the prediction phase. Cross-Origin Resource Sharing (CORS) is enabled securely to allow the frontend to interact with the backend from different origins during development.

In a production deployment, HTTPS must be enabled to secure traffic, and environment variables should be used to store sensitive credentials and model paths. Rate-limiting, authentication tokens (JWT), and additional layers of validation can be added if the system is scaled up.

I. Deployment Considerations

The system is modular and can be deployed either locally or in the cloud. The backend Flask server can be hosted using Gunicorn or uWSGI behind an Nginx reverse proxy. Model files are large and must be handled using Git LFS or external hosting (Google Drive, AWS S3). The frontend can be hosted using services like Netlify, Vercel, or served statically via Nginx.

During deployment, care must be taken to:

- Limit memory usage by batching predictions
- Monitor inference latency
- Use secure, tokenized API communication
- Prevent file upload abuse via size and type checks

VI. Experimental Results Evaluation

A. Evaluation Metrics

The model was evaluated on the combined test set using the following metrics:

- **Accuracy:** Overall correct predictions / total predictions.
- **Precision, Recall:** Per-class metrics computed for imbalance sensitivity.
- **F1-Score (Macro):** Average F1 across all classes.
- **Confusion Matrix:** Visual representation of class-level performance.

B. Model Performance Comparison

We compared four architectures across validation datasets:

ConvNeXt-Large outperformed all baselines in both accuracy and F1-score. The inclusion of TTA and SWA helped stabilize performance in minority classes like fear, disgust, and sadness.

C. Confusion Matrix

Most confusion occurred between ”neutral” and ”sad” classes, highlighting the need for more samples and perhaps multi-label classification in future iterations.

Model	Dataset	Training Acc.	Validation Acc.
EfficientNet-B0	FER2013	65.52%	60.00%
EfficientNet-B4	AffectNet	52.55%	54.76%
ConvNeXt Small	AffectNet	66.96%	58.75%
ConvNeXt Base	AffectNet	73.39%	60.25%
ConvNeXt Large	AffectNet	73.36%	60.25%
ConvNeXt Large	RAF-DB	99.10%	85.82%
CNN-LSTM (static)	FER2013	62.50%	57.25%
ConvNeXt-Large	Combined	**89.65%**	**81.74%**

TABLE II
Model Comparison across Datasets

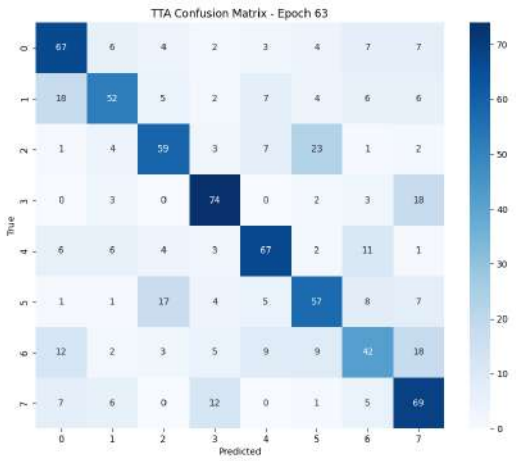


Fig. 19. Confusion Matrix – ConvNeXt-Large on Combined Validation Set

D. Ablation Study

We conducted an ablation study to isolate the effect of:

- Removing SWA → performance dropped by 1.3%.
- Disabling Mixup → increased overfitting, especially in early epochs.
- Skipping TTA → test-time predictions fluctuated more across runs.

E. Discussion

These experiments confirm that emotion classification models benefit significantly from augmentation and stability strategies. ConvNeXt-Large, in particular, proved robust to dataset noise and imbalance. The model generalizes well across three datasets, suggesting potential for deployment in real-world scenarios.

VII. Web App Functionalities

A. Key Modules Demonstration

The proposed system has been implemented as a full-stack web application that enables users to perform real-time emotion detection through facial expression analysis. The application integrates a React-based front-end, a Flask backend, and pre-trained deep learning models for emotion inference.

The application consists of the following key modules:

- **User Authentication Module:** Enables users to register and log in securely using bcrypt-hashed credentials. The frontend includes real-time form validation, and the backend interfaces with a MySQL database to manage user records.

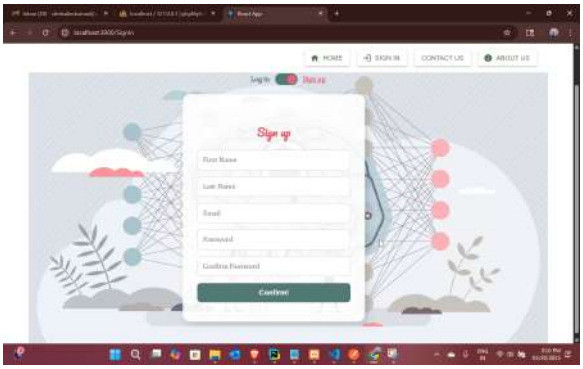


Fig. 20. Sign-up and login interface with form validation.

- **Dashboard Interface:** Upon successful login, users are redirected to a dynamic dashboard. This interface provides access to the webcam and allows users to record a short video clip capturing facial expressions.

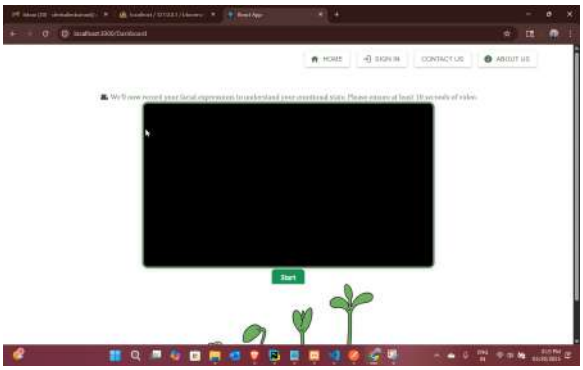


Fig. 21. Dashboard interface with access to webcam recording.

- **Webcam Integration:** The frontend uses the MediaRecorder API to capture video data directly from the user's webcam. The recorded clip is stored as a Blob and transmitted to the backend via an HTTP POST request.
- **Emotion Prediction Engine:** The Flask backend extracts frames from the uploaded video, preprocesses them, and passes them to a trained deep learning model. The model returns a probability distribution over emotion classes, and the most frequently occurring emotion is returned as the final output.
- **Results Display:** The predicted emotion and, optionally, its probability distribution are displayed on the dashboard UI. The results are presented in an intuitive and visually engaging format.

This modular design allows seamless communication between client and server components and ensures that users can

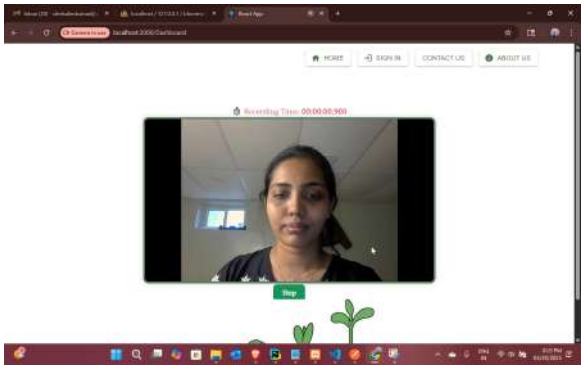


Fig. 22. Webcam recording interface using MediaRecorder API.

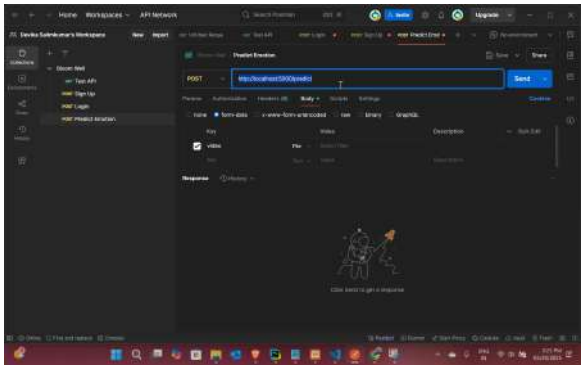


Fig. 23. Backend prediction result displayed in response to video input.

interact with the system in real-time. The system was tested on multiple devices and browsers and was able to consistently provide accurate and timely predictions across different facial inputs

To evaluate the practical implementation of the proposed system, a fully functional web application was developed using React.js and Flask. The system enables users to register, log in, record real-time webcam video, and receive emotion predictions from a deep learning model hosted on the backend. This section presents screenshots demonstrating each major feature and UI component.

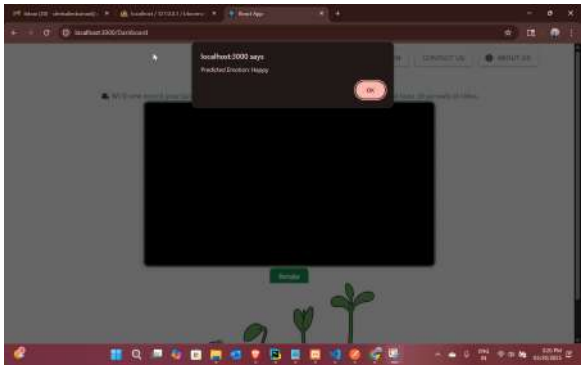


Fig. 24. Predicted emotion results presented on the dashboard.

B. Main Page and Home page

C. Login and SignUp Page

D. Dashboard

E. Contact Us

F. Database: MySql

G. Postman for Testing APIs

VIII. Conclusion and Future Work

A. Summary of Findings

This project presents a comprehensive, real-time system for detecting potential indicators of mental disorders through facial emotion analysis. Leveraging deep learning models such as EfficientNet-B4, Vision Transformer (ViT), and CNN-LSTM, trained on diverse and widely adopted datasets like AffectNet and RAF-DB, the system achieved high accuracy in classifying core human emotions. The web-based implementation—developed using React for the frontend and Flask for the backend—facilitates seamless user interaction, real-time webcam integration, and backend-based emotion prediction using a trained neural network. The modular architecture allows flexible deployment, while the inclusion of user authentication and a clean dashboard design enhances usability.

B. Limitations and Future Enhancements

While the system demonstrates high accuracy and responsiveness, several limitations and implementation challenges were encountered:

- **Model Deployment Constraints:** Due to hardware limitations and GitHub file size restrictions, model inference had to be managed externally or optimized using Git LFS and temporary file handling.
- **Inference Latency:** The system currently processes video uploads sequentially on the backend, which introduces some latency—especially when handling longer videos or large frame batches.
- **Hardware Dependency:** Real-time predictions require significant computational power; local CPU-based inference may not scale well without GPU acceleration or cloud deployment.
- **Demographic Generalization:** Like many emotion datasets, AffectNet and RAF-DB may not fully represent global diversity in facial expressions, leading to potential bias in prediction across demographics.
- **Unimodal Limitation:** The current system relies solely on facial expressions. It does not yet incorporate other psychological indicators such as speech, tone, or textual cues from user input.
- **Lack of Clinical Validation:** While the system works technically, it has not yet been clinically validated for use in mental health diagnostics or screening.

C. Future Directions

To enhance the system's performance, scalability, and clinical utility, the following future improvements are proposed:

Fig. 25. Landing page and it redirects to the home page on clicking the navigation button HOME

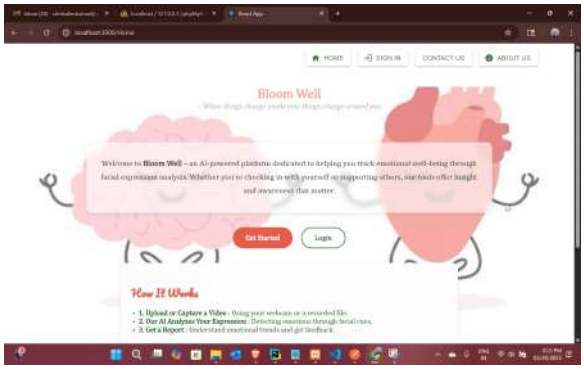


Fig. 26. Home Page

Fig. 27. A small description on the home page about how the web app works

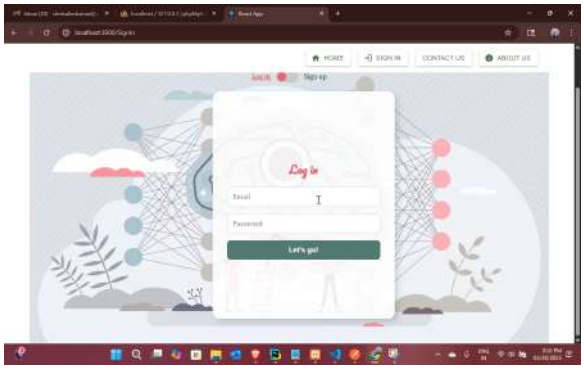


Fig. 28. Login Page

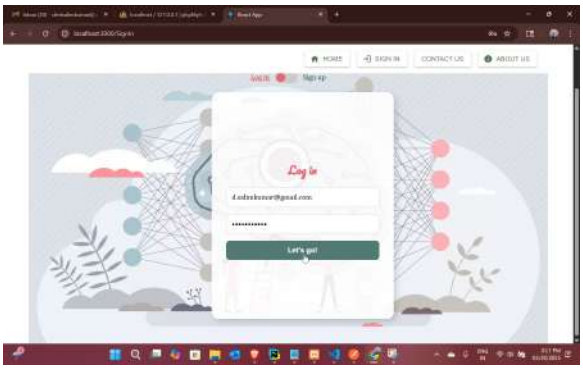


Fig. 29. Login using as not an existing user

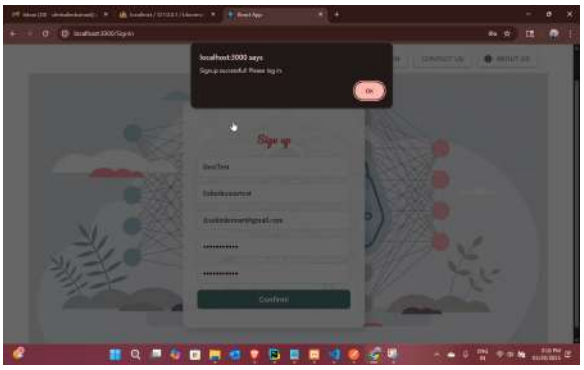


Fig. 33. Signed up as a new user

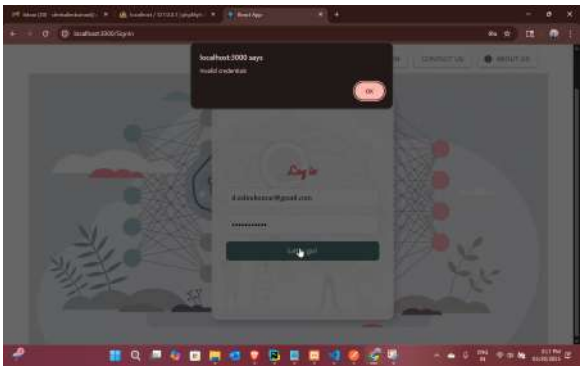


Fig. 30. Invalid credentials is displayed after checking with in database

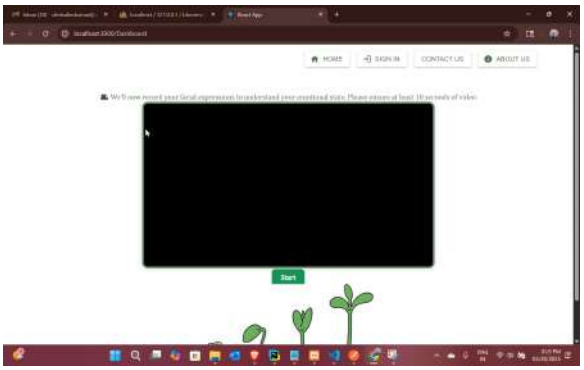


Fig. 34. Dashboard work flow:1

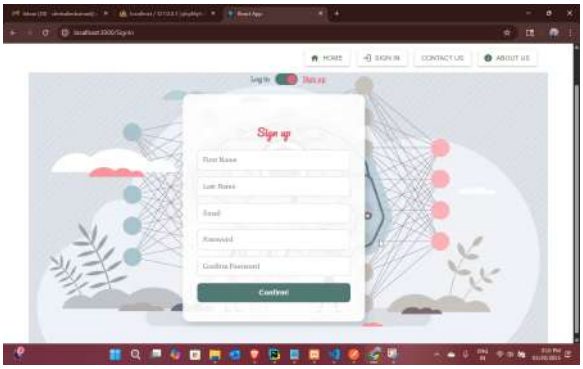


Fig. 31. Sign up module

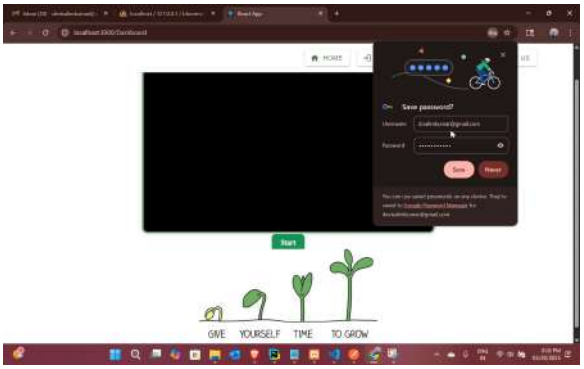


Fig. 35. Dashboard work flow:2

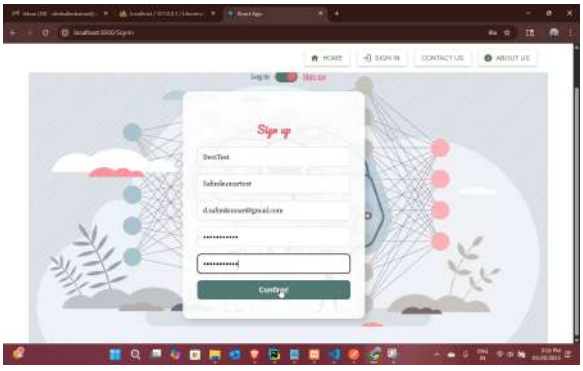


Fig. 32. Signing up as a new user

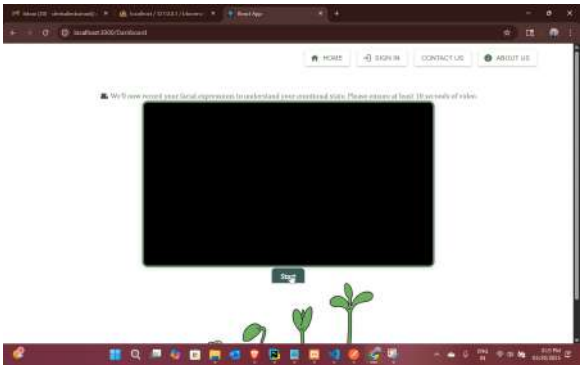


Fig. 36. Dashboard work flow:3

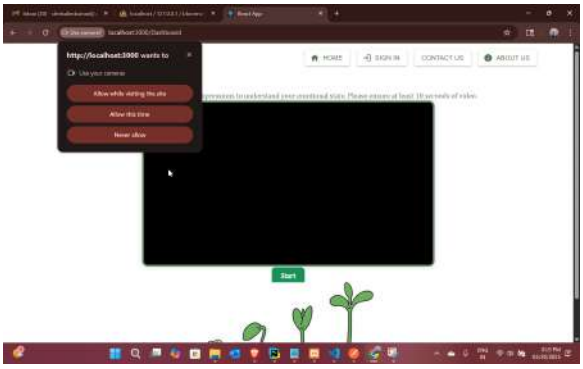


Fig. 37. Dashboard work flow:4

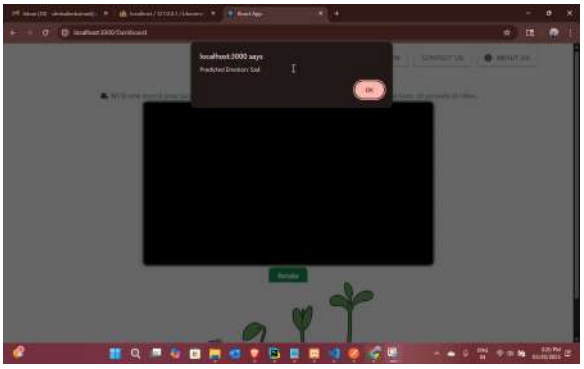


Fig. 41. Dashboard work flow:8

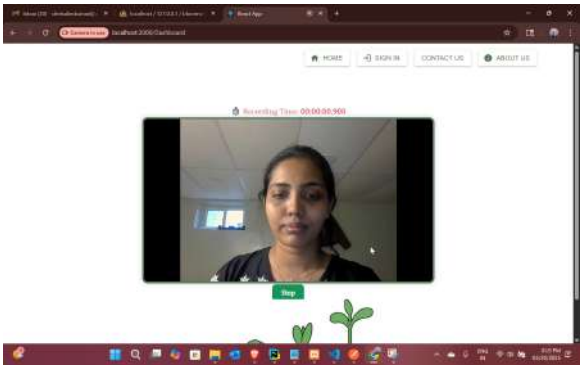


Fig. 38. Dashboard work flow:5

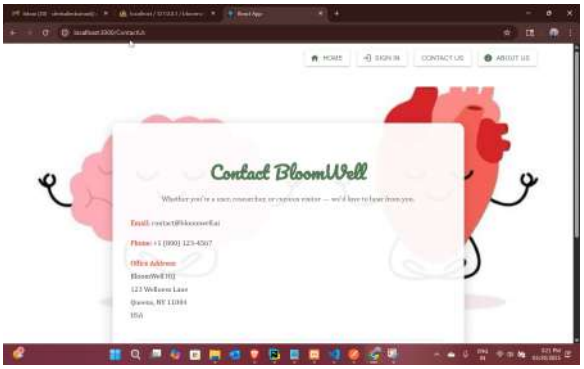


Fig. 42. Contact Us work flow:1

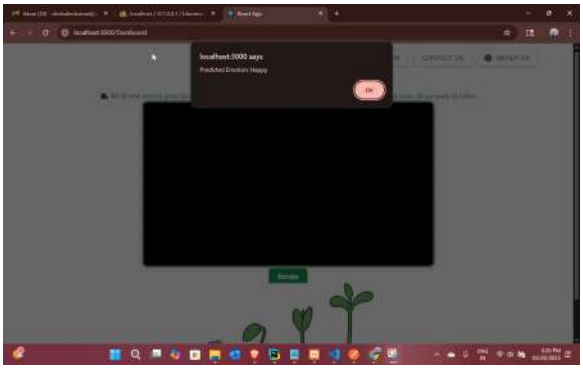


Fig. 39. Dashboard work flow:6

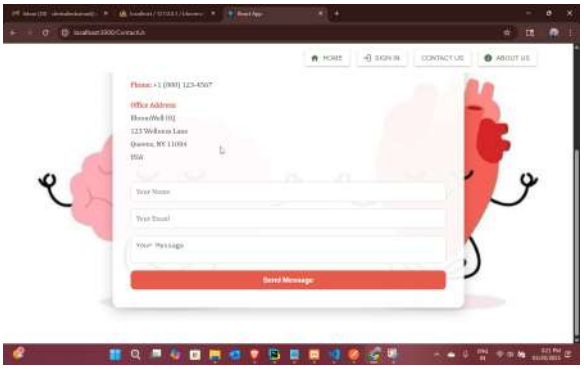


Fig. 43. Contact Us work flow:2

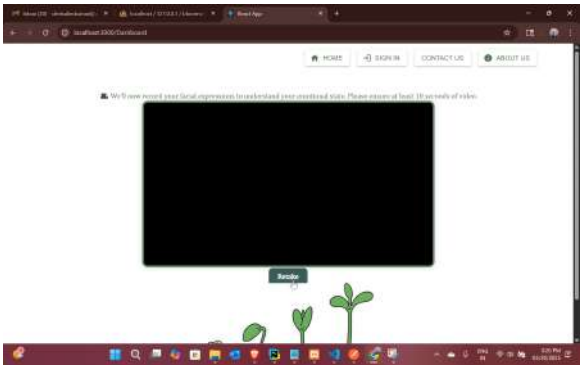


Fig. 40. Dashboard work flow:7

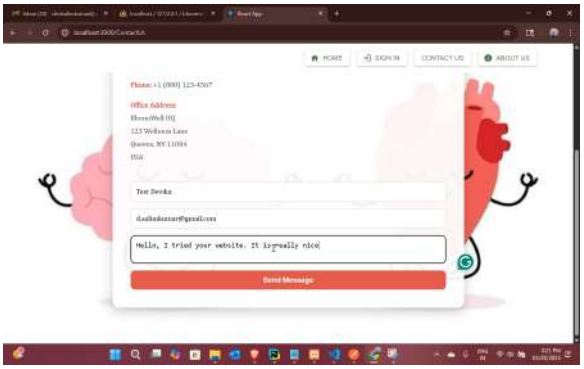


Fig. 44. Contact Us work flow:3

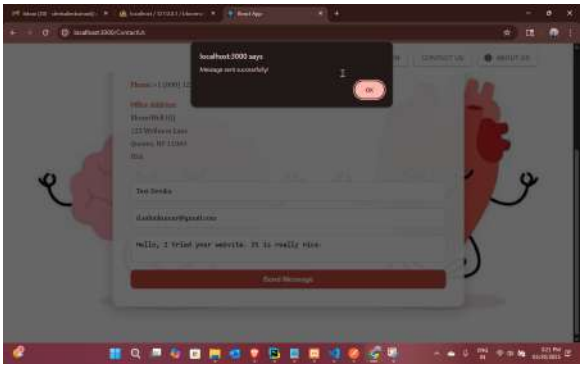


Fig. 45. Contact Us work flow:4

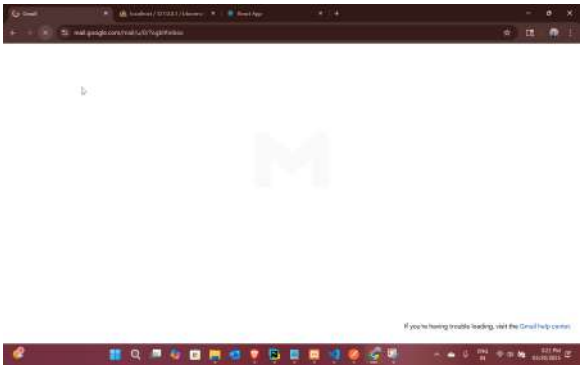


Fig. 46. Contact Us work flow:5

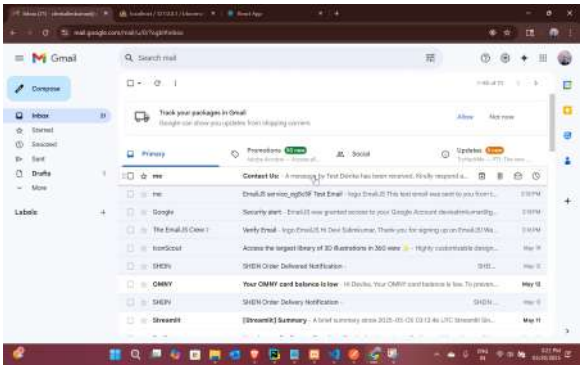


Fig. 47. Contact Us work flow:6

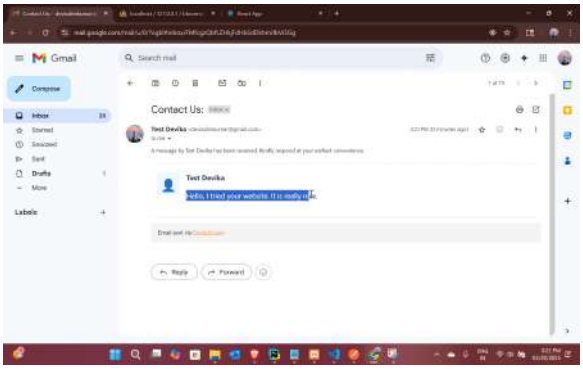


Fig. 48. Contact Us work flow:7

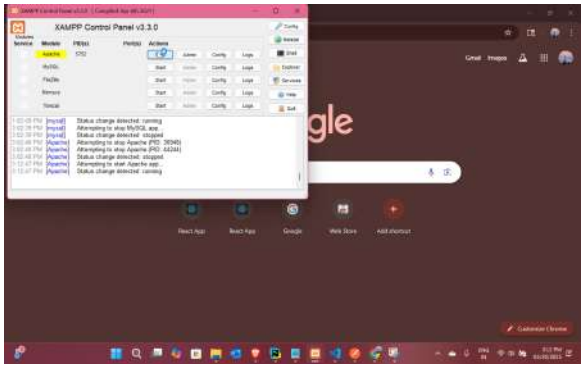


Fig. 49. Database fig:1

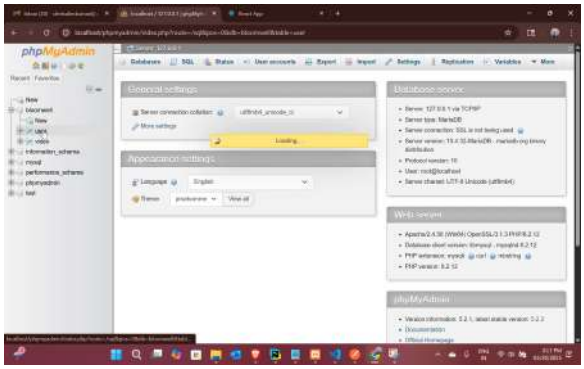


Fig. 50. Database fig:2

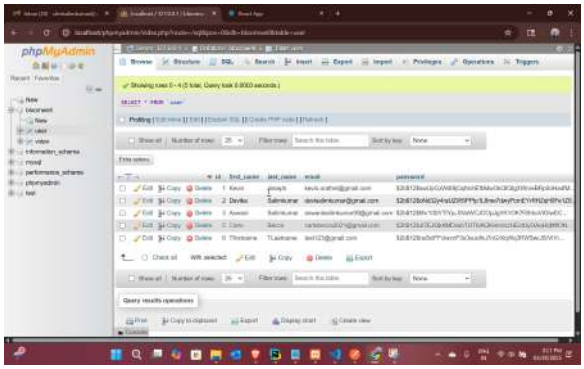


Fig. 51. Database fig:3

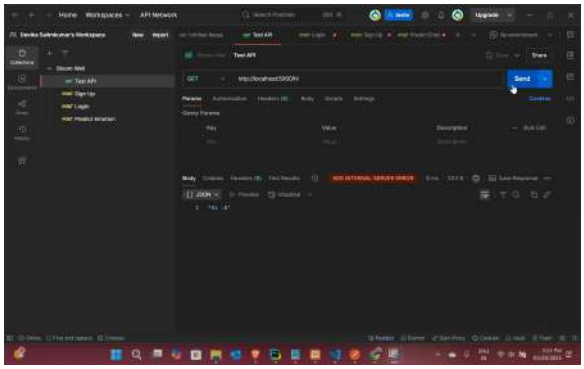


Fig. 52. Postman testing:1

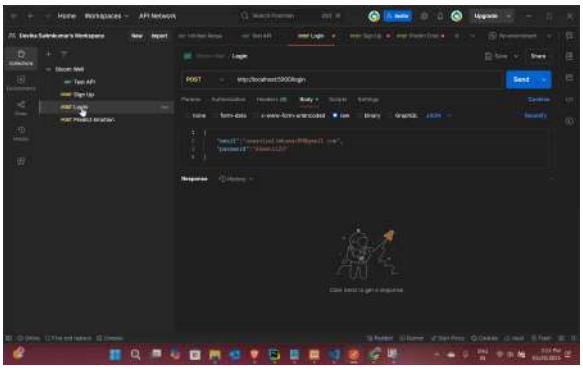


Fig. 53. Postman testing:2

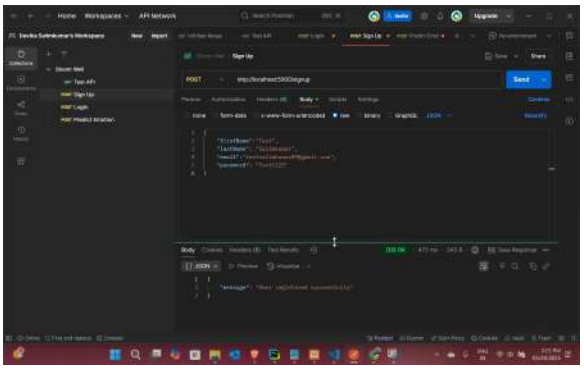


Fig. 57. Postman testing:6

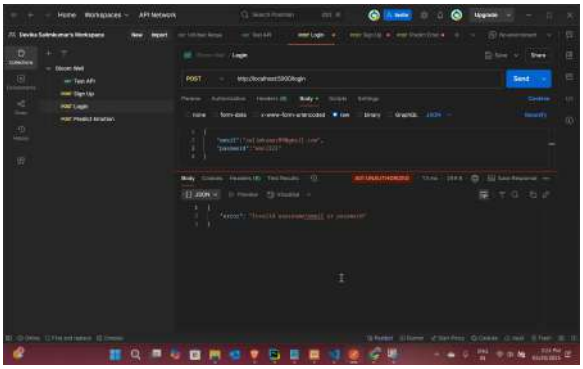


Fig. 54. Postman testing:3

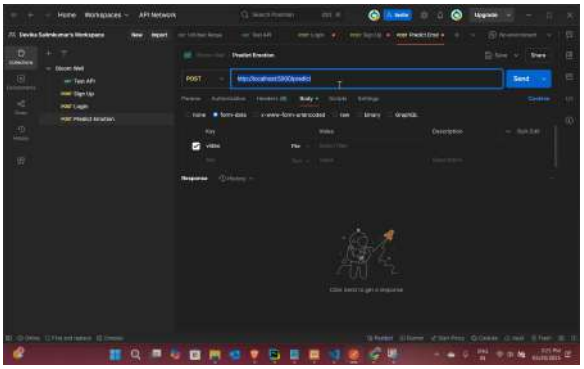


Fig. 58. Postman testing:7

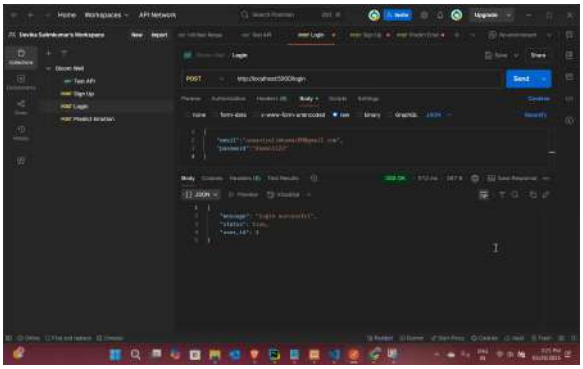


Fig. 55. Postman testing:4

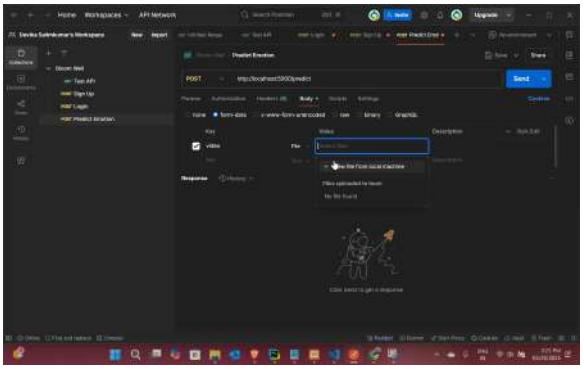


Fig. 59. Postman testing:8

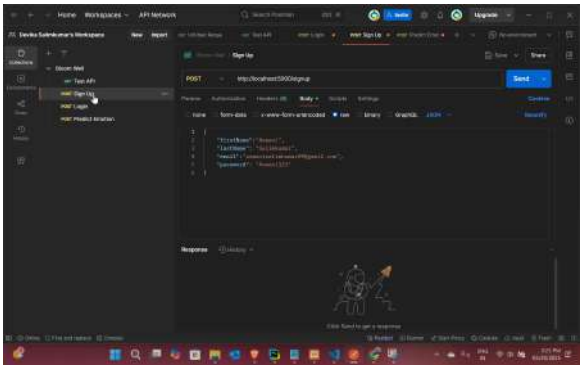


Fig. 56. Postman testing:5

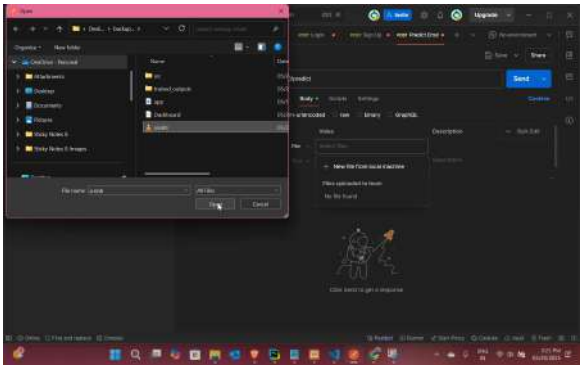


Fig. 60. Postman testing:9

- **Multimodal Emotion Analysis:** Future versions of the system will incorporate voice analysis and natural language processing to combine audio, text, and facial data for more accurate mental state inference.
- **Cloud-based Inference:** Deploying the emotion prediction API on cloud platforms such as Google Cloud Functions, AWS Lambda, or Azure ML will enable scalable, low-latency inference.
- **Edge Deployment:** Optimizing the model using ONNX or TensorRT will allow deployment on edge devices such as smartphones, tablets, or Raspberry Pi for offline or field use.
- **Temporal Emotion Tracking:** Integrating CNN-LSTM or ConvLSTM models will allow analysis of emotion changes over time rather than static predictions from individual frames.
- **Explainability and Interpretability:** Adding explainable AI (XAI) techniques such as Grad-CAM, SHAP, or LIME will improve trust and transparency, particularly in clinical and research settings.
- **Professional Validation:** Collaborations with psychologists and mental health practitioners will be pursued to validate the system in real-world settings and assess its applicability in early-stage mental health screening.

In conclusion, this system bridges deep learning technology with practical healthcare needs, laying the groundwork for an accessible, non-invasive, and scalable tool for early detection and monitoring of emotional and psychological well-being.

ACKNOWLEDGMENT

We would like to sincerely thank our faculty mentor, **Professor Ihab Darwish** <ihab.darwish60@login.cuny.edu>, for his invaluable guidance throughout the semester. We also acknowledge the support provided by The City College of New York, our department faculty, and compute resources via Kaggle and Google Colab Pro. Tools such as GitHub, Firebase, and Overleaf contributed significantly to our development and collaboration workflows.

The source code and training pipeline are available at:

- for model training: https://github.com/Capstone-Project-CCNY/AI-Driven-Mental-Disorder-Detection-/tree/main
- for web application codes: https://github.com/Capstone-Project-CCNY/AI-Driven-Mental-Disorder-Detection-/tree/frontend-dev

REFERENCES

[1] Z. Liu, H. Mao, C.-Y. Wu, et al., "A ConvNet for the 2020s," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 11976–11986.

[2] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 6105–6114.

[3] A. Mollahosseini, D. Chan, and M. H. Mahoor, "AffectNet: A database for facial expression, valence, and arousal computing in the wild," *IEEE Trans. Affective Comput.*, vol. 10, no. 1, pp. 18–31, 2019.

[4] S. Li and W. Deng, "Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 2584–2593.

[5] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[6] P. Izmailov et al., "Averaging weights leads to wider optima and better generalization," in *Proc. Conf. Uncertainty Artif. Intell. (UAI)*, 2018.

[7] D. Wang et al., "Test-Time Training with Self-Supervision for Generalization under Distribution Shifts," in *Proc. NeurIPS*, 2020.

[8] A. Ahmed et al., "AI-Driven Mental Disorders Categorization from Social Media: A Deep Learning Pre-Screening Framework," in *Proc. Int. Conf. Mach. Intell. Smart Innov. (ICMISI)*, 2024, doi: 10.1109/ICMISI61517.2024.10580665.

[9] B. Singh et al., "A Hybrid Learning Architecture for Mental Disorder Detection Using Emotion Recognition," *IEEE Access*, vol. 12, pp. 91410–91425, 2024.

[10] S. Ramesh et al., "Machine Learning-Based Detection of PTSD in Mental Health," in *Proc. Int. Conf. Electron. Sustain. Commun. Syst. (ICESC)*, 2023.

[11] M. Kaya et al., "Multi-task Learning on Mental Disorder Detection Using Social Media Posts," in *Proc. Int. Symp. Innov. Approaches Smart Technol. (ISAS)*, 2024.

[12] V. Patel et al., "Improving Mental Health Assessments: CNN-SVM for Depression Detection," in *Proc. Int. Conf. Adv. Technol. (ICONAT)*, 2024.

APPENDIX A
CODE SNIPPETS

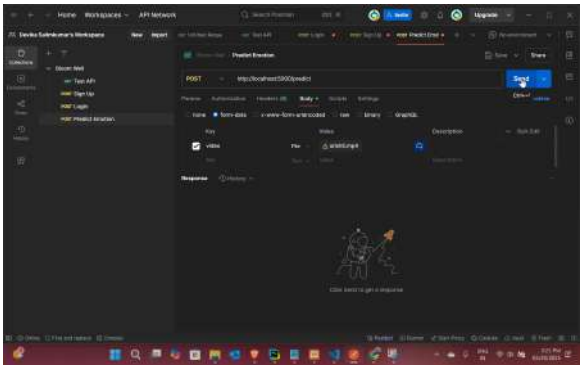


Fig. 61. Postman testing:10

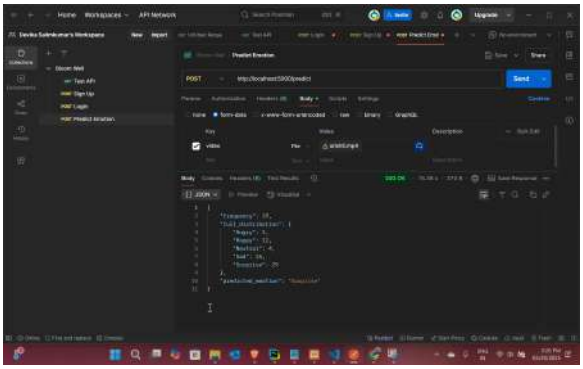


Fig. 62. Postman testing:11

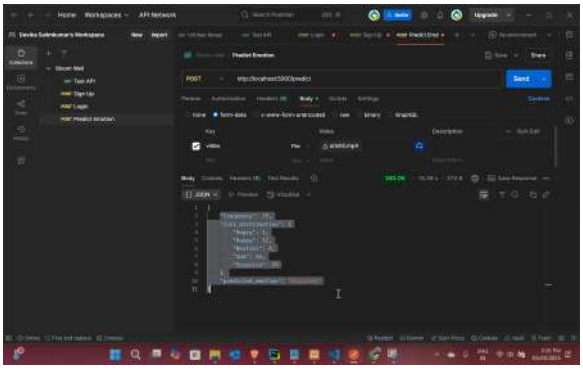


Fig. 63. Postman testing:12

A. Model Loading and Inference (Flask API)

```
import torch
from torchvision import transforms
from model import ConvNeXtLarge

model = ConvNeXtLarge(pretrained=False)
model.load_state_dict(torch.load("model_final.pt", map_location='cpu'))
model.eval()

def predict(image_tensor):
    with torch.no_grad():
        outputs = model(image_tensor.unsqueeze(0))
        _, predicted = torch.max(outputs, 1)
        return predicted.item()
```

B. Back-end code: app.py

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import model_from_json
from tensorflow.keras.preprocessing.image import img_to_array
import cv2
import numpy as np
import mysql.connector
import os
from werkzeug.utils import secure_filename
from collections import Counter
import bcrypt

app = Flask(__name__)

db_config = {
    'host': '127.0.0.1',
    'user': 'root',
    'password': '',
    'database': 'bloomwell',
    'port': '3306'
}

app.config['UPLOAD_FOLDER'] = 'uploads'
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

# Emotion labels (adjust if your model uses a different set)
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']

def get_db_connection():
    return mysql.connector.connect(**db_config)

# Load model from JSON and weights
def load_emotion_model():
    with open('trained_outputs/model.json', 'r') as json_file:
        model_json = json_file.read()
    model = model_from_json(model_json)
    model.load_weights('trained_outputs/model.weights.h5')
    return model

model = load_emotion_model()
```

```
def preprocess_frame(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    face = cv2.resize(gray, (48, 48))
    face = img_to_array(face)
    face = np.expand_dims(face, axis=0)
    face /= 255.0  # Normalize
    return face

@app.route('/predict', methods=['POST'])
def predict_emotion():
    if 'video' not in request.files:
        return jsonify({'error': 'No video file provided'}), 400

    video = request.files['video']
    filename = secure_filename(video.filename)
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    video.save(filepath)

    cap = cv2.VideoCapture(filepath)
    emotion_counts = Counter()
    frame_skip = 5  # Analyze every 5th frame to reduce processing load
    frame_index = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        if frame_index % frame_skip == 0:
            try:
                face_input = preprocess_frame(frame)
                prediction = model.predict(face_input, verbose=0)[0]
                predicted_emotion = emotion_labels[np.argmax(prediction)]
                emotion_counts[predicted_emotion] += 1
            except Exception as e:
                print(f"Error processing frame {frame_index}: {e}")
        frame_index += 1

    cap.release()
    os.remove(filepath)  # Clean up uploaded file

    if not emotion_counts:
        return jsonify({'error': 'No valid frames for prediction'}), 500

    # Get emotion with highest frequency
    most_common_emotion = emotion_counts.most_common(1)[0]
    return jsonify({
        'predicted_emotion': most_common_emotion[0],
        'frequency': most_common_emotion[1],
        'full_distribution': dict(emotion_counts)
    })

@app.route("/hi", methods=["GET"])
def hi():
    return jsonify("Hi :D"), 500

@app.route('/login', methods=['POST'])
```

```
def login():
    data = request.get_json()
    email = data.get('email')
    password = data.get('password')

    if not email or not password:
        return jsonify({'error': 'Email and password are required'}), 400

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    query = "SELECT * FROM user WHERE email = %s"
    cursor.execute(query, (email,))
    user = cursor.fetchone()

    cursor.close()
    conn.close()

    if user and bcrypt.checkpw(password.encode('utf-8'), user['password'].encode('utf-8')):
        return jsonify({'message': 'Login successful', 'user_id': user['id']})
    else:
        return jsonify({'error': 'Invalid username/email or password'}), 401

@app.route('/signup', methods=['POST'])
def register():
    data = request.get_json()
    firstName = data.get('firstName')
    lastName = data.get('lastName')
    email = data.get('email')
    password = data.get('password')

    if not email or not password:
        return jsonify({'error': 'All fields are required'}), 400

    password_hash = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        cursor.execute("INSERT INTO user (first_name, last_name, email, password) VALUES (%s,
                       (firstName, lastName, email, password_hash))
        conn.commit()
        return jsonify({'message': 'User registered successfully'})
    except mysql.connector.IntegrityError:
        return jsonify({'error': 'Email already exists'}), 409
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    app.run(debug=True)
```

C. Front-end code: App.js

```
    import React, { useState } from "react";
import {
```

```
  BrowserRouter as Router,
  Routes,
  Route,
  useNavigate,
} from "react-router-dom";
import Home from "./components/Home";
import SignUp from "./components/SignUp";
import ContactUS from "./components/ContactUs";
import About from "./components/AboutUs";
import Dashboard from "./components/Dashboard";

import {
  AppBar,
  Toolbar,
  Button,
  Box,
  Typography,
  Container,
} from "@mui/material";
import { Home as HomeIcon, Info, Login } from "@mui/icons-material";
import "./App.css";

function App() {
  const [showAbout, setShowAbout] = useState(false);
  const navigate = useNavigate(); // for navigation

  return (
    <div className="app-container">
      {/* Material UI Navigation Bar */}
      <AppBar
        position="fixed"
        sx={{ backgroundColor: "white", boxShadow: "none" }}
      >
        <Container maxWidth="lg">
          <Toolbar sx={{ justifyContent: "flex-end" }}>
            <Box sx={{ display: "flex", gap: 2 }}>
              <Button
                onClick={() => navigate("/Home")}
                variant="contained"
                color="inherit"
                startIcon={<HomeIcon />}
                sx={{
                  color: "#537955",
                  backgroundColor: "transparent",
                  "&:hover": {
                    color: "#e95d4d",
                    backgroundColor: "transparent",
                  },
                }}
              >
                Home
              </Button>
              <Button
                onClick={() => navigate("/SignIn")}
                variant="contained"
                color="inherit"
```

```
            startIcon={<Login />}
            sx={{
              color: "#537955",
              backgroundColor: "transparent",
              "&:hover": {
                color: "#e95d4d",
                backgroundColor: "transparent",
              },
            }}
          >
            Sign In
          </Button>
          <Button
            onClick={() => navigate("/ContactUs")}
            variant="contained"
            sx={{
              color: "#537955",
              backgroundColor: "transparent",
              "&:hover": {
                color: "#e95d4d",
                backgroundColor: "transparent",
              },
            }}
          >
            Contact Us
          </Button>
          <Button
            onClick={() => navigate("/AboutUs")}
            variant="contained"
            onMouseEnter={() => setShowAbout(true)}
            onMouseLeave={() => setShowAbout(false)}
            color="inherit"
            startIcon={<Info />}
            sx={{
              color: "#537955",
              backgroundColor: "transparent",
              "&:hover": {
                color: "#e95d4d",
                backgroundColor: "transparent",
              },
            }}
          >
            About Us
          </Button>
        </Box>
      </Toolbar>
    </Container>
  </AppBar>

  {/* Main Content */}
  {showAbout && (
    <Box
      onMouseEnter={() => setShowAbout(true)}
      onMouseLeave={() => setShowAbout(false)}
      sx={{
        position: "absolute",
```

```
      top: "100px",
      left: 0,
      width: "280px",
      backgroundColor: "#e95d4d",
      borderRight: "4px solid #e95d4d",
      padding: "1rem 1.2rem",
      boxShadow: "4px 0 8px rgba(0,0,0,0.1)",
      zIndex: 1000,
      transition: "all 0.3s ease-in-out",
    }}
  >
    <Typography
      variant="h6"
      sx={{ color: "#ffff", fontWeight: "bold", fontFamily: "Pacifico" }}
    >
      Our Vision
    </Typography>
    <Typography variant="body2" sx={{ mt: 1, color: "#ffff" }}>
      To leverage AI for early detection of emotional and mental disorders
      to create healthier societies.
    </Typography>

    <Typography
      variant="h6"
      sx={{ mt: 2, color: "#ffff", fontWeight: "bold" }}
    >
      Our Mission
    </Typography>
    <Typography variant="body2" sx={{ mt: 1, color: "#ffff" }}>
      We aim to empower individuals through awareness and access to
      cutting-edge emotion analysis tools built on deep learning.
    </Typography>
  </Box>
)}

<Container sx={{ mt: 12, textAlign: "center" }}>
  <Routes>
    <Route
      path="/"
      element={
        <div className="content-container">
          <h1 className="title">Bloom Well</h1>
          <h4 className="tagline">
            "When things change inside you, things change around you"
          </h4>
          <img
            src="/background.jpg"
            alt="Mind Blooming"
            className="bg-image"
          />
        </div>
      }
    />
      <Route path="/Home" element={<Home/>}/>
      <Route path="/SignIn" element={<SignUp/>}/>
      <Route path="/Dashboard" element={<Dashboard />} />
```

```
            <Route path="/ContactUs" element={<ContactUS/>}/>
            <Route path="/AboutUs" element={<About/>}/>
        </Routes>
      </Container>
    </div>
  );
}

export default App;
```

D. Front-end code: App.css

```
    .app-container {
  overflow: hidden;
  display: flex;
  min-height: 100vh;
  background-color: white;
}

.content-container {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.title {
  font-family: "Pacifico", cursive;
  color: #e95d4d;
  font-size: 4.5rem;
  margin-bottom: 0.2rem;
}

.tagline {
  font-family: "Pacifico", cursive;
  color: #537955;
  font-size: 1.5rem;
  align-self: self-end;
  margin-top: 0.3rem;
  margin-left: 30%;
}

.bg-image {
  width: 450px;
  height: auto;
  border: none;
  box-shadow: none;
  margin-top: 0.2rem;
}
```

E. Front-end code: Home.js

```
   import React, { useRef, useState } from "react";
import { useNavigate } from "react-router-dom";
import "../styles/Home.css";

function Home() {
  const navigate = useNavigate();
  const [isRecording, setIsRecording] = useState(false);
```

```
const videoRef = useRef(null);

const handleCapture = async () => {
  setIsRecording(true);
  const stream = await navigator.mediaDevices.getUserMedia({ video: true });
  videoRef.current.srcObject = stream;

  const mediaRecorder = new MediaRecorder(stream);
  const chunks = [];

  mediaRecorder.ondataavailable = (e) => chunks.push(e.data);

  mediaRecorder.onstop = async () => {
    const blob = new Blob(chunks, { type: "video/webm" });
    const formData = new FormData();
    formData.append("video", blob, "capture.webm");

    const response = await fetch("http://127.0.0.1:5000/predict", {
      method: "POST",
      body: formData,
    });

    const result = await response.json();
    alert("Predicted Emotion: " + result.emotion);

    stream.getTracks().forEach((track) => track.stop());
    setIsRecording(false);
  };

  mediaRecorder.start();
  setTimeout(() => {
    mediaRecorder.stop();
  }, 5000); // Record for 5 seconds
};

return (
  <div className="home-wrapper">
    <div className="overlay" />

    {/*  Custom hamburger icon */}
    <div className="custom-hamburger" onClick={() => navigate("/")}>
      <input type="checkbox" id="checkbox" />
      <label htmlFor="checkbox" className="toggle">
        <div className="bars" id="bar1"></div>
        <div className="bars" id="bar2"></div>
        <div className="bars" id="bar3"></div>
      </label>
    </div>

    <div className="home-heading-container">
      <h1 className="home-heading">Bloom Well</h1>
      <p className="home-tagline">
        { When things change inside you, things change around you
      </p>
    </div>
```

```
      <div className="home-main-content">
        <section className="home-intro fade-in">
          <p>
            Welcome to <strong>Bloom Well</strong> { an AI-powered platform
            dedicated to helping you track emotional well-being through facial
            expression analysis. Whether you're checking in with yourself or
            supporting others, our tools offer insight and awareness that
            matter.
          </p>
        </section>

        <section className="home-buttons fade-in">
          <button
            className="btn-get-started"
            onClick={() => navigate("/SignIn", { state: { flip: "signup" } })}
          >
            Get Started
          </button>
          <button
            className="btn-signin"
            onClick={() => navigate("/SignIn", { state: { flip: "login" } })}
          >
            Login
          </button>
        </section>

        <section className="home-steps fade-in">
          <h3>How It Works</h3>
          <ul>
            <li>
              <strong>1. Upload or Capture a Video</strong> : Using your webcam
              or a recorded file.
            </li>
            <li>
              <strong>2. Our AI Analyzes Your Expression</strong> : Detecting
              emotions through facial cues.
            </li>
            <li>
              <strong>3. Get a Report</strong> : Understand emotional trends and
              get feedback.
            </li>
          </ul>
        </section>

        <section className="home-benefits fade-in">
          <h3>Why Choose Bloom Well?</h3>
          <ul>
            <li> Private, secure, and real-time analysis</li>
            <li> Easy to use, no sign-in needed to start</li>
            <li> Designed for awareness and mental wellness</li>
          </ul>
        </section>
      </div>
    </div>
  );
}
```

```
export default Home;
```

F. Front-end code: Home.css

```
    .home-wrapper {
  background-image: url("../assets/home-bg.jpg");
  background-size: cover;
  background-position: center;
  background-attachment: fixed;
  min-height: 100vh;
  position: relative;
  font-family: Cambria, serif;
  overflow-x: hidden;
}

.overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(255, 255, 255, 0.457);
  z-index: 1;
}

.home-heading {
  font-size: 1.8rem;
  color: #e95d4d;
  margin: 0;
}

.home-tagline {
  font-family: Cambria, serif;
  font-style: italic;
  color: #537955;
  font-size: 1rem;
  margin: 0;
}

.home-main-content {
  position: relative;
  z-index: 2;
  text-align: center;
  max-width: 900px;
  margin: 70px auto 40px auto;
  padding: 0 1.5rem;
}

.home-intro {
  font-size: 1.1rem;
  color: #3d3d3d;
  margin-bottom: 2rem;
  line-height: 1.7;
  background: rgba(255, 255, 255, 0.6);
  padding: 1rem 1.5rem;
  border-radius: 12px;
```

```
  backdrop-filter: blur(5px);
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

.home-buttons {
  margin-bottom: 2rem;
  display: flex;
  justify-content: center;
  gap: 1.2rem;
}

.btn-get-started,
.btn-signin {
  padding: 12px 28px;
  font-size: 1rem;
  border-radius: 25px;
  border: none;
  cursor: pointer;
  transition: all 0.3s ease;
  font-family: Cambria, serif;
}

.btn-get-started {
  background-color: #e95d4d;
  color: white;
}

.btn-get-started:hover {
  background-color: #d2473e;
  transform: scale(1.05);
}

.btn-signin {
  background-color: white;
  border: 2px solid #537955;
  color: #537955;
}

.btn-signin:hover {
  background-color: #537955;
  color: white;
  transform: scale(1.05);
}

.home-steps,
.home-benefits {
  background: #fdfdfd;
  border-radius: 12px;
  padding: 1.5rem;
  margin: 2rem auto;
  text-align: left;
  max-width: 700px;
  box-shadow: 0px 2px 8px rgba(0, 0, 0, 0.08);
  font-family: Cambria, serif;
}
```

```
.home-steps h3,
.home-benefits h3 {
  color: #e95d4d;
  font-size: 1.4rem;
  margin-bottom: 1rem;
  font-family: "Pacifico", sans-serif;
}

.home-steps ul,
.home-benefits ul {
  padding-left: 1rem;
  font-size: 1rem;
  color: #266229;
  list-style: disc;
}

.fade-in {
  animation: fadeInUp 1.2s ease-in-out;
}

@keyframes fadeInUp {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

.custom-hamburger {
  position: fixed;
  top: 20px;
  left: 20px;
  z-index: 1002;
}

#checkbox {
  display: none;
}

.toggle {
  position: relative;
  width: 40px;
  height: 40px;
  cursor: pointer;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  gap: 10px;
  transition-duration: 0.3s;
}

.bars {
```

```
  width: 100%;
  height: 4px;
  background-color: rgb(76, 189, 151);
  border-radius: 5px;
  transition-duration: 0.3s;
}

#checkbox:checked + .toggle .bars {
  margin-left: 13px;
}

#checkbox:checked + .toggle #bar2 {
  transform: rotate(135deg);
  margin-left: 0;
  transform-origin: center;
  transition-duration: 0.3s;
}

#checkbox:checked + .toggle #bar1 {
  transform: rotate(45deg);
  transition-duration: 0.3s;
  transform-origin: left center;
}

#checkbox:checked + .toggle #bar3 {
  transform: rotate(-45deg);
  transition-duration: 0.3s;
  transform-origin: left center;
}
```

G. Front-end code: SingUp.js

```
    import React, { useState } from "react";
import { useLocation } from "react-router-dom";
import { useNavigate } from "react-router-dom";
import "../styles/SignUp.css";
import backgroundImage from "../assets/bg.jpeg";

function SignUp() {
  const navigate = useNavigate();
  const location = useLocation();
  const [isSignUp, setIsSignUp] = useState(location.state?.flip !== "login");
  const [message, setMessage] = useState(""); // For message text
  const [messageType, setMessageType] = useState(""); // "success" or "error"

  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [signupEmail, setSignupEmail] = useState("");
  const [signupPassword, setSignupPassword] = useState("");
  const [confirmSignupPassword, setConfirmSignupPassword] = useState("");
  const [error, setError] = useState("");
  const handleLoginSubmit = (e) => {
    e.preventDefault();

    if (!email.trim() || !password.trim()) {
```

```
    setError("Email and password are required");
    return;
  }

  fetch(`http://localhost:5000/login`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      email: email,
      password: password,
    }),
  })
    .then((response) => response.json())
    .then((data) => {
      if (data.status === true) {
        setError("");
        navigate("/Dashboard");
      } else {
        setError("Invalid credentials");
        alert("Invalid credentials");
      }
    })
    .catch((error) => {
      setError("An error occurred");
      alert("An error occurred");
    });

  setFirstName("");
  setLastName("");
  setSignupEmail("");
  setSignupPassword("");
  setConfirmSignupPassword("");
};

const handleSignupSubmit = (e) => {
  e.preventDefault();

  if (
    !firstName.trim() ||
    !signupEmail.trim() ||
    !signupPassword.trim() ||
    !confirmSignupPassword.trim()
  ) {
    setError("All fields are required");
    alert("All fields are required");
    return;
  }

  if (signupPassword !== confirmSignupPassword) {
    setError("Passwords do not match");
    alert("Passwords do not match");
    return;
  }
```

```
  fetch(`http://localhost:5000/signup`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      email: signupEmail,
      password: signupPassword,
      firstName: firstName,
      lastName: lastName,
    }),
  })
    .then((response) => response.json())
    .then((data) => {
      alert("Signup successful! Please log in.");
      setFirstName("");
      setLastName("");
      setSignupEmail("");
      setSignupPassword("");
      setConfirmSignupPassword("");
    })
    .catch((error) => {
      setError("An error occurred");
      alert("An error occurred");
    });
};

return (
  <div
    className="signup-container"
    style={{
      backgroundImage: `url(${backgroundImage})`,
      backgroundSize: "cover",
      backgroundPosition: "center",
      backgroundRepeat: "no-repeat",
      backgroundAttachment: "fixed",
      minHeight: "100vh",
    }}
  >
    <div className="toggle-wrapper">
      <span
        onClick={() => setIsSignUp(false)}
        className={!isSignUp ? "active" : ""}
      >
        Log in
      </span>
      <label className="switch">
        <input
          type="checkbox"
          checked={isSignUp}
          onChange={() => setIsSignUp(!isSignUp)}
        />
        <span className="slider" />
      </label>
      <span
        onClick={() => setIsSignUp(true)}
```

```
    className={isSignUp ? "active" : ""}
  >
    Sign up
  </span>
</div>

<div className={`flip-card ${isSignUp ? "flipped" : ""}`}>
  <div className="flip-card-inner">
    {/* Log in */}
    <div className="flip-card-front">
      <h2>Log in</h2>
      <form onSubmit={handleLoginSubmit}>
        <input
          type="email"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <button type="submit">Let's go!</button>
      </form>
    </div>

    {/* Sign up */}
    <div className="flip-card-back">
      <h2>Sign up</h2>
      <form onSubmit={handleSignupSubmit}>
        <input
          type="text"
          placeholder="First Name"
          value={firstName}
          onChange={(e) => setFirstName(e.target.value)}
        />
        <input
          type="text"
          placeholder="Last Name"
          value={lastName}
          onChange={(e) => setLastName(e.target.value)}
        />
        <input
          type="email"
          placeholder="Email"
          value={signupEmail}
          onChange={(e) => setSignupEmail(e.target.value)}
        />
        <input
          type="password"
          placeholder="Password"
          value={signupPassword}
          onChange={(e) => setSignupPassword(e.target.value)}
        />
```

```
              <input
                type="password"
                placeholder="Confirm Password"
                value={confirmSignupPassword}
                onChange={(e) => setConfirmSignupPassword(e.target.value)}
              />
              <button type="submit">Confirm!</button>
            </form>
          </div>
        </div>
      </div>
    </div>
  );
}

export default SignUp;
```

H. Front-end code: SingUp.css

```
    @import url("https://fonts.googleapis.com/css2?family=Pacifico&display=swap");

body {
  margin: 0;
  padding: 0;
  font-family: Cambria, serif;
  background-size: auto;
  background-repeat: no-repeat;
  min-height: 100vh;
}

.signup-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: flex-start;
  min-height: 100vh;
  padding-top: 60px; /* moves the card up */
  box-sizing: border-box;
}

.toggle-wrapper {
  display: flex;
  align-items: center;
  gap: 12px;
  margin-bottom: 20px;
  font-weight: bold;
  color: #537975;
}

.toggle-wrapper span {
  cursor: pointer;
  transition: color 0.3s;
}

.toggle-wrapper .active {
  text-decoration: underline;
```

```
  color: #e86574;
}

.switch {
  position: relative;
  width: 50px;
  height: 26px;
}

.switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

.slider {
  position: absolute;
  cursor: pointer;
  background-color: #ccc;
  border-radius: 26px;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  transition: 0.4s;
}

.slider::before {
  content: "";
  position: absolute;
  height: 20px;
  width: 20px;
  left: 3px;
  bottom: 3px;
  background-color: #e86574;
  border-radius: 50%;
  transition: 0.4s;
}

input:checked + .slider {
  background-color: #537975;
}

input:checked + .slider::before {
  transform: translateX(24px);
}

/* Flip Card */
.flip-card {
  width: 340px;
  height: 440px;
  perspective: 1000px;
}

.flip-card-inner {
  width: 100%;
```

```
  height: 100%;
  transition: transform 0.8s;
  transform-style: preserve-3d;
  position: relative;
}

.flipped .flip-card-inner {
  transform: rotateY(180deg);
}

.flip-card-front,
.flip-card-back {
  position: absolute;
  width: 100%;
  height: 100%;
  backface-visibility: hidden;
  background: rgba(255, 255, 255, 0.92);
  border-radius: 16px;
  box-shadow: 6px 6px 14px rgba(0, 0, 0, 0.15);
  padding: 2rem;
  display: flex;
  flex-direction: column;
  justify-content: center;
}

.flip-card-back {
  transform: rotateY(180deg);
}

h2 {
  text-align: center;
  font-family: "Pacifico", cursive;
  color: #e86574;
  margin-bottom: 1.2rem;
}

form {
  display: flex;
  flex-direction: column;
  gap: 15px;
}

input {
  padding: 12px;
  border-radius: 8px;
  border: 1px solid #ccc;
  font-size: 1rem;
  font-family: Cambria, serif;
}

button {
  background-color: #537975;
  color: white;
  border: none;
  padding: 12px;
  border-radius: 8px;
```

```
  font-size: 1rem;
  cursor: pointer;
  font-weight: bold;
  transition: 0.3s ease;
}

button:hover {
  background-color: #537975;
}
```

I. Front-end code: Dashboard.js

```
    import React, { useRef, useState, useEffect } from "react";
import "../styles/Dashboard.css";
import growImage from "../assets/dashboard.jpg";

function Dashboard() {
  const videoRef = useRef(null);
  const mediaRecorderRef = useRef(null);
  const [isRecording, setIsRecording] = useState(false);
  const [videoBlob, setVideoBlob] = useState(null);
  const [timer, setTimer] = useState(0);
  const timerIntervalRef = useRef(null);

  const formatTime = (ms) => {
    const date = new Date(ms);
    return (
      date.toISOString().substr(11, 8) +
      "." +
      String(ms % 1000).padStart(3, "0")
    );
  };

  const startCamera = async () => {
    const stream = await navigator.mediaDevices.getUserMedia({ video: true });
    videoRef.current.srcObject = stream;
    mediaRecorderRef.current = new MediaRecorder(stream);

    const chunks = [];
    mediaRecorderRef.current.ondataavailable = (e) => chunks.push(e.data);

    mediaRecorderRef.current.onstop = () => {
      const blob = new Blob(chunks, { type: "video/webm" });
      setVideoBlob(blob);
      stream.getTracks().forEach((track) => track.stop());
      clearInterval(timerIntervalRef.current);
    };

    mediaRecorderRef.current.start();
    setIsRecording(true);
    setTimer(0);

    timerIntervalRef.current = setInterval(() => {
      setTimer((prev) => prev + 100);
    }, 100);
  };
```

```
const stopCamera = () => {
  if (mediaRecorderRef.current) {
    mediaRecorderRef.current.stop();
  }
  setIsRecording(false);

  // Add a slight delay to ensure blob is fully set
  setTimeout(() => {
    sendVideoForPrediction();
  }, 500);
};

const retake = () => {
  setVideoBlob(null);
  setIsRecording(false);
  setTimer(0); // Reset the timer
  startCamera(); // Restart everything
};

const sendVideoForPrediction = async () => {
  if (!videoBlob) {
    alert("No video recorded yet.");
    return;
  }

  const formData = new FormData();
  formData.append("video", videoBlob, "input_video.webm");

  try {
    const response = await fetch("http://localhost:5000/predict", {
      method: "POST",
      body: formData,
    });

    const result = await response.json();

    if (response.ok) {
      alert(
        `Predicted Emotion: ${result.predicted_emotion}`
      );
      console.log("Full distribution:", result.full_distribution);
    } else {
      alert(result.error || "Prediction failed.");
    }
  } catch (error) {
    console.error("Error during prediction:", error);
    alert("An error occurred while predicting emotion.");
  }
};

useEffect(() => {
  return () => clearInterval(timerIntervalRef.current); // Cleanup on unmount
}, []);

return (
  <div className="dashboard-container">
```

```
      <p className="note">
         We'll now record your facial expressions to understand your emotional
        state. Please ensure at least 10 seconds of video.
      </p>

      {isRecording && (
        <p className="timer">
           Recording Time: <strong>{formatTime(timer)}</strong>
        </p>
      )}

      <video ref={videoRef} autoPlay muted className="video-frame" />

      <div className="button-group">
        {!isRecording && !videoBlob && (
          <button onClick={startCamera}>Start</button>
        )}
        {isRecording && <button onClick={stopCamera}>Stop</button>}
        {videoBlob && !isRecording && <button onClick={retake}>Retake</button>}
      </div>

      <div className="growth-image-container">
        <img src={growImage} alt="Grow Message" className="growth-image" />
      </div>
    </div>
  );
}

export default Dashboard;
```

J. Front-end code: Dashboard.css

```
    .dashboard-container {
    min-height: 100vh;
    max-height: 100vh;
    overflow: hidden;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: flex-start;
    margin: 0;
    font-family: Cambria, serif;
    box-sizing: border-box;
}

.note {
    color: #537955;
    font-size: 1rem;
    margin-top: 0.5rem;
    margin-bottom: 1rem;
    text-align: center;
    white-space: nowrap; /* Prevent line breaks */
    overflow: hidden;
    text-overflow: ellipsis; /* Optional: Adds ... if it overflows */
    max-width: 100vw; /* Make sure it spans across */
}
```

```
.timer {
    font-size: 1.1rem;
    color: #e86574;
    margin-bottom: 1rem;
}

.video-frame {
    width: 640px;
    height: 360px; /* 16:9 ratio */
    max-width: 90%;
    margin-top: -15px;
    border: 4px solid #537955;
    border-radius: 12px;
    background-color: black;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
}

.button-group {
    margin-top: 10px;
    display: flex;
    justify-content: center;
    gap: 1rem;
}

.button-group button {
    background-color: #189457;
    color: white;
    border: none;
    padding: 10px 20px;
    margin-top: -8px;
    font-size: 1rem;
    border-radius: 8px;
    cursor: pointer;
    font-family: Cambria, serif;
    transition: 0.3s ease;
}

.button-group button:hover {
    background-color: #3c5e59;
}

.growth-image-container {
  width: 100%;
  display: flex;
  justify-content: center;
}

.growth-image {
    width: 100%;
    max-width: 640px;
    aspect-ratio: 16 / 5;
    object-fit: contain;
    margin-top: -5px;
}
```

K. Front-end code: ContactUs.js

```
    import React from "react";
import("../styles/ContactUs.css");

function ContactUs() {
  return (
    <div style={{ textAlign: "center", marginTop: "100px" }}>
      <h2>Contact Us</h2>
      <p>You can reach us at contact@bloomwell.ai</p>
    </div>
  );
}

export default ContactUs;
```

L. Data Load and Preprocessing

```
#LOADING DATASETS

import zipfile
!kaggle datasets download -d msambare/fer2013
with zipfile.ZipFile("fer2013.zip", "r") as zip_ref:
    zip_ref.extractall("fer2013")



# === Install kagglehub (if not already installed) ===
!pip install -q kagglehub

import kagglehub
import shutil
import os

# === Download dataset ===
path = kagglehub.dataset_download("shuvoalok/raf-db-dataset")

print("KaggleHub Downloaded to:", path)

# === Copy it to /content/raf-db ===
destination = "/content/raf-db"
if not os.path.exists(destination):
    shutil.copytree(path, destination)
    print(f"Dataset copied to: {destination}")
else:
    print(f"Dataset already exists at: {destination}")


!kaggle datasets download -d thienkhonghoc/affectnet

!unzip -q /content/affectnet.zip -d /content/affectnet > /dev/null 2>&1

from google.colab import files

# Upload the zip file
uploaded = files.upload()  # browse and select wild_images.zip
```

```
import zipfile
import os

# Unzip it to a known path (e.g., /content/wild_images)
zip_path = "/content/wild_test_images.zip"
extract_path = "/content/wild_test_images"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f"Extracted to: {extract_path}")

!kaggle datasets download -d tapakah68/facial-emotion-recognition
!unzip facial-emotion-recognition.zip -d facial_emotion_recognition

import zipfile
import os

# Unzip it to a known path (e.g., /content/wild_images)
zip_path = "/content/wild_labeled_augmented.zip"
extract_path = "/content/wild_labeled_augmented"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f"Extracted to: {extract_path}")



#DATA PROCESSING


    # Ensure GPU is Used & Optimize Memory Usage
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    for gpu in gpus:
        tf.config.experimental.set_memory_growth(gpu, True)
    tf.config.set_visible_devices(gpus[0], 'GPU')

# Defining Image Properties
IMG_SIZE = 224
BATCH_SIZE = 32  # Larger batch size to better utilize GPU



# Data Augmentation (Same as Before)
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
```

```
    brightness_range=[0.5, 1.5],
    fill_mode="nearest"
)


# Load Data
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2, p
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=2, pin_
```

M. Model Training and Validation

```
#  Load ConvNeXt-Large Model
model = models.convnext_large(weights=models.ConvNeXt_Large_Weights.IMAGENET1K_V1)

#  Modify Classifier (Fix LayerNorm Shape)
model.classifier = nn.Sequential(
    nn.AdaptiveAvgPool2d(1),  #  Convert (B, 1536, 7, 7) → (B, 1536, 1, 1)
    nn.Flatten(),  #  Convert (B, 1536, 1, 1) → (B, 1536)
    nn.LayerNorm(1536),  #  Now works correctly
    nn.Dropout(0.5),
    nn.Linear(1536, 512),
    nn.ReLU(),
    nn.BatchNorm1d(512),
    nn.Dropout(0.4),
    nn.Linear(512, 8)
)

#  Optimize Model Memory Usage
model = torch.compile(model)
model = model.to(device)

#  Define Loss, Optimizer & Scheduler
criterion = nn.CrossEntropyLoss(weight=weights, label_smoothing=0.1)
optimizer = optim.AdamW(model.parameters(), lr=5e-5, weight_decay=1e-6)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=15, eta_min=1e-7)

#  Mixed Precision Training
scaler = GradScaler()

#  Train for 30 Epochs
print("\n Training ConvNeXt-Large with Fixes...\n")

for epoch in range(1, 31):
    model.train()
    running_loss, correct_train, total_train = 0.0, 0, 0

    optimizer.zero_grad()

    for i, (images, labels) in enumerate(train_loader):
        if images is None or labels is None:
            continue

        images, labels = images.to(device), labels.to(device)
```

```
        with autocast():
            outputs = model(images)
            loss = criterion(outputs, labels) / accumulation_steps

        scaler.scale(loss).backward()

        if (i + 1) % accumulation_steps == 0:
            scaler.step(optimizer)
            scaler.update()
            optimizer.zero_grad()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct_train += (predicted == labels).sum().item()
        total_train += labels.size(0)

    train_accuracy = 100 * correct_train / total_train
    scheduler.step()

    # Validation Phase
    model.eval()
    correct_val, total_val = 0, 0
    with torch.no_grad():
        for images, labels in val_loader:
            if images is None or labels is None:
                continue

            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            correct_val += (predicted == labels).sum().item()
            total_val += labels.size(0)

    val_accuracy = 100 * correct_val / total_val

    print(f"Epoch [{epoch}/30], Loss: {running_loss:.4f}, Train Acc: {train_accuracy:.2f}%, Va

    if epoch % 5 == 0:
        torch.save(model.state_dict(), f"affectnet_convnext_large_epoch{epoch}.pt")

torch.save(model.state_dict(), "affectnet_convnext_large_final.pt")
print("\n Training complete! Final model saved.")

# TRAINING LOOP WITH ALL 3 DATASETS

# === Model Setup ===
model = convnext_large(weights=None)
model.classifier = nn.Sequential(
    nn.Flatten(),
    nn.LayerNorm(1536),
    nn.Linear(1536, 512),
    nn.ReLU(),
    nn.Linear(512, 7)
)
```

```
# === Load Previous Best Checkpoint ===
checkpoint_path = "/content/model_highest_train0.81acc0.8392.pt"
model.load_state_dict(torch.load(checkpoint_path, map_location="cuda"), strict=False)
print(" Loaded checkpoint:", checkpoint_path)

# === Unfreeze deeper layers ===
for name, param in model.features[6:].named_parameters():
    param.requires_grad = True
model.to("cuda")

# === Optimizer with per-layer LR ===
backbone_params = [p for n, p in model.named_parameters() if "features" in n and p.requires_gra
classifier_params = model.classifier.parameters()
optimizer = torch.optim.AdamW([
    {"params": backbone_params, "lr": 1e-5},
    {"params": classifier_params, "lr": 1e-4}
], weight_decay=1e-2)

criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
scheduler = CosineAnnealingWarmRestarts(optimizer, T_0=5, T_mult=2)
scaler = GradScaler()

# === Training Loop ===
best_val_acc = 0
patience = 3
patience_counter = 0

for epoch in range(20):
    print(f"\n--- Epoch {epoch+1} ---")
    model.train()
    total_loss, correct, total = 0, 0, 0

    for inputs, targets in tqdm(train_loader, desc="Training", leave=False):
        inputs, targets = inputs.to("cuda"), targets.to("cuda")
        optimizer.zero_grad()
        with autocast(device_type="cuda"):
            outputs = model(inputs)
            loss = criterion(outputs, targets)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        total_loss += loss.item()
        correct += (outputs.argmax(1) == targets).sum().item()
        total += targets.size(0)

    train_loss = total_loss / len(train_loader)
    train_acc = correct / total
    scheduler.step()

    print(f"Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f}")

    # === Validation ===
    model.eval()
    y_true, y_pred = [], []
    val_loss = 0
```

```
    with torch.no_grad():
        for inputs, targets in tqdm(val_loader, desc="Validation", leave=False):
            inputs, targets = inputs.to("cuda"), targets.to("cuda")
            with autocast(device_type="cuda"):
                outputs = model(inputs)
                loss = criterion(outputs, targets)
            preds = torch.argmax(outputs, 1)
            y_pred.extend(preds.cpu().numpy())
            y_true.extend(targets.cpu().numpy())
            val_loss += loss.item()

    val_loss /= len(val_loader)
    val_acc = accuracy_score(y_true, y_pred)
    print(f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save(model.state_dict(), "best_model.pt")
        print(" New best model saved.")
        patience_counter = 0
    else:
        patience_counter += 1
        print(f"Patience: {patience_counter}/{patience}")
        if patience_counter >= patience:
            print(" Early stopping triggered.")
            break


#EVALUATION

from sklearn.metrics import classification_report, confusion_matrix

labels = [0, 1, 2, 3, 4, 5, 6]
names = ["Anger", "Disgust", "Fear", "Happiness", "Sadness", "Surprise", "Neutral"]

print(classification_report(
    y_true, y_pred,
    labels=labels,
    target_names=names,
    zero_division=0
))

conf_mat = confusion_matrix(y_true, y_pred, labels=labels)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues",
            xticklabels=names, yticklabels=names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

N. Model Training - Model Test

```
import os
import torch
```

```
import numpy as np
from PIL import Image
from tqdm import tqdm
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from torchvision.models import convnext_large
import torch.nn as nn
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
GLOBAL_LABELS = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
GLOBAL_LABEL_MAP = {label: i for i, label in enumerate(GLOBAL_LABELS)}

# Dataset for FER2013 or folder-based data
class FolderEmotionDataset(Dataset):
    def __init__(self, root_dir, transform, label_map=GLOBAL_LABEL_MAP):
        self.samples = []
        self.transform = transform
        for label in os.listdir(root_dir):
            class_dir = os.path.join(root_dir, label)
            if not os.path.isdir(class_dir): continue
            label_idx = label_map.get(label.lower())
            if label_idx is None: continue
            for file in os.listdir(class_dir):
                if file.endswith((".jpg", ".png")):
                    self.samples.append((os.path.join(class_dir, file), label_idx))

    def __len__(self): return len(self.samples)

    def __getitem__(self, idx):
        path, label = self.samples[idx]
        image = Image.open(path).convert("RGB")
        return transform(image), label

# Dataset for RAF-DB using CSV
class RAFDBFromCSV(Dataset):
    def __init__(self, img_dir, csv_file, transform):
        self.img_dir = img_dir
        self.transform = transform
        self.samples = []
        raf_map = {1: 6, 2: 2, 3: 1, 4: 3, 5: 5, 6: 0, 7: 4}
        df = pd.read_csv(csv_file)
        for _, row in df.iterrows():
            filename = row['image']
            label = raf_map[row['label']]
            for subfolder in map(str, range(1, 8)):
                full_path = os.path.join(img_dir, subfolder, filename)
                if os.path.isfile(full_path):
                    self.samples.append((full_path, label))
                    break

    def __len__(self): return len(self.samples)

    def __getitem__(self, idx):
```

```
        path, label = self.samples[idx]
        image = Image.open(path).convert("RGB")
        return transform(image), label

# Transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

# Load model
model = convnext_large(weights=None)
model.classifier = nn.Sequential(
    nn.Flatten(),
    nn.LayerNorm(1536),
    nn.Linear(1536, 512),
    nn.ReLU(),
    nn.Linear(512, 7)
)
model.load_state_dict(torch.load("/content/model_unfrozen_epoch_9.pt", map_location=device))
model.to(device)
model.eval()

# Evaluation function
def evaluate(name, dataset):
    loader = DataLoader(dataset, batch_size=32, shuffle=False)
    all_preds, all_labels = [], []
    with torch.no_grad():
        for imgs, labels in tqdm(loader, desc=f"Evaluating {name}"):
            outputs = model(imgs.to(device))
            preds = outputs.argmax(1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.numpy())
    acc = np.mean(np.array(all_preds) == np.array(all_labels))
    print(f"\n{name} Accuracy: {acc:.4f}")
    print(classification_report(all_labels, all_preds, target_names=GLOBAL_LABELS, zero_divisi
    cm = confusion_matrix(all_labels, all_preds, labels=list(range(7)))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GLOBAL_LABELS)
    disp.plot(cmap="Blues", xticks_rotation=45)
    plt.title(f"{name} Confusion Matrix")
    plt.grid(False)
    plt.tight_layout()
    plt.show()

# Evaluate FER2013
fer_dataset = FolderEmotionDataset("/content/fer2013/test", transform)
evaluate("FER2013", fer_dataset)

# Evaluate RAF-DB
raf_dataset = RAFDBFromCSV("/content/raf-db/DATASET/test", "/content/raf-db/test_labels.csv",
evaluate("RAF-DB", raf_dataset)


import os
import torch
```

```
import numpy as np
from PIL import Image
from tqdm import tqdm
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from torchvision.models import convnext_large
import torch.nn as nn

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
GLOBAL_LABELS = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
GLOBAL_LABEL_MAP = {label: i for i, label in enumerate(GLOBAL_LABELS)}

# === Dataset Loader ===
class FolderEmotionDataset(Dataset):
    def __init__(self, root_dir, transform, label_map=GLOBAL_LABEL_MAP):
        self.samples = []
        self.transform = transform
        for label in os.listdir(root_dir):
            class_dir = os.path.join(root_dir, label)
            if not os.path.isdir(class_dir): continue
            label_idx = label_map.get(label.lower())
            if label_idx is None: continue
            for file in os.listdir(class_dir):
                if file.endswith((".jpg", ".png")):
                    self.samples.append((os.path.join(class_dir, file), label_idx))

    def __len__(self): return len(self.samples)

    def __getitem__(self, idx):
        path, label = self.samples[idx]
        image = Image.open(path).convert("RGB")
        return transform(image), label

# === Image Transform ===
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

# === Load Model ===
model = convnext_large(weights=None)
model.classifier = nn.Sequential(
    nn.Flatten(),
    nn.LayerNorm(1536),
    nn.Linear(1536, 512),
    nn.ReLU(),
    nn.Linear(512, 7)
)
model.load_state_dict(torch.load("/content/model_unfrozen_epoch_9.pt", map_location=device))
model.to(device)
model.eval()

# === Evaluation Function ===
```

```
def evaluate(name, dataset):
    loader = DataLoader(dataset, batch_size=32, shuffle=False)
    all_preds, all_labels = [], []
    with torch.no_grad():
        for imgs, labels in tqdm(loader, desc=f"Evaluating {name}"):
            outputs = model(imgs.to(device))
            preds = outputs.argmax(1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.numpy())
    acc = np.mean(np.array(all_preds) == np.array(all_labels))
    print(f"\n{name} Accuracy: {acc:.4f}")
    print(classification_report(all_labels, all_preds, target_names=GLOBAL_LABELS, zero_divisi
    return all_labels, all_preds

# === Run Evaluation on FER2013 ===
fer_dataset = FolderEmotionDataset("/content/fer2013/test", transform)
all_labels, all_preds = evaluate("FER2013", fer_dataset)

# === Plot Confusion Matrix ===
cm = confusion_matrix(all_labels, all_preds, labels=list(range(7)))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GLOBAL_LABELS)
disp.plot(cmap="Blues", xticks_rotation=45)
plt.title("FER2013 Confusion Matrix")
plt.tight_layout()
plt.show()

# === Plot Per-Class F1 Scores ===
report = classification_report(all_labels, all_preds, target_names=GLOBAL_LABELS, output_dict='
f1_scores = [report[label]['f1-score'] for label in GLOBAL_LABELS]

plt.figure(figsize=(8, 4))
plt.bar(GLOBAL_LABELS, f1_scores, color='skyblue')
plt.title("FER2013 Per-Class F1 Scores")
plt.ylim(0, 1)
plt.grid(True, axis='y')
plt.ylabel("F1 Score")
plt.tight_layout()
plt.show()


import torch.nn.functional as F

image = captured_img.convert("RGB")
image = transform(image).unsqueeze(0).to(device)

with torch.no_grad():
    output = model(image)
    probs = F.softmax(output, dim=1)
    pred_idx = output.argmax(1).item()
    pred_label = GLOBAL_LABELS[pred_idx]
    confidence = probs[0][pred_idx].item()

print(f" Webcam Prediction  {pred_label} ({confidence:.2f})")

from IPython.display import display, Javascript
from google.colab.output import eval_js
```

```
from base64 import b64decode
import cv2
import numpy as np
from PIL import Image
import io

# Capture a webcam image
def capture_image():
    js = Javascript('''
        async function capture() {
            const stream = await navigator.mediaDevices.getUserMedia({ video: true });
            const video = document.createElement('video');
            video.srcObject = stream;
            await video.play();

            const canvas = document.createElement('canvas');
            canvas.width = 640;
            canvas.height = 480;
            canvas.getContext('2d').drawImage(video, 0, 0);

            stream.getTracks().forEach(track => track.stop());
            return canvas.toDataURL('image/jpeg');
        }
        capture();
    ''')
    display(js)
    data = eval_js('capture()')
    header, encoded = data.split(',', 1)
    img_bytes = b64decode(encoded)
    return Image.open(io.BytesIO(img_bytes))

captured_img = capture_image()
captured_img.save("webcam.jpg")
print(" Image captured!")


from torchvision.models import convnext_large
import torch.nn as nn
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = convnext_large(weights=None)
model.classifier = nn.Sequential(
    nn.Flatten(),
    nn.LayerNorm(1536),
    nn.Linear(1536, 512),
    nn.ReLU(),
    nn.Linear(512, 7)
)
model.load_state_dict(torch.load("/content/model_unfrozen_epoch_9.pt", map_location=device))
model.to(device)
model.eval()
example_input = torch.randn(1, 3, 224, 224).to(device)
traced_model = torch.jit.trace(model, example_input)
```

```
# Save the TorchScript model
traced_model.save("/content/emotion_model_ts.pt")
print(" Model exported to TorchScript as 'emotion_model_ts.pt'")
```