Project Title: Real-time Facial Emotion Recognition System

Author: Regino Balogo Jr.

Section: BSCS 3-A
Date: 05/23/2025

I. Project Overview & Objectives

A. Overview

This project develops a real-time facial emotion recognition system capable of detecting and classifying human emotions (angry, disgust, fear, happy, neutral, sad, surprise) from a live camera feed. The system leverages deep learning techniques, specifically a Convolutional Neural Network (CNN) trained on the FER-2013 dataset, integrated into an interactive web application built with Streamlit. OpenCV is utilized for camera access and efficient face detection.

B. Objectives

The primary objectives of this project were:

1. **Dataset Preparation:** Efficiently load and preprocess the FER-2013 dataset, incorporating data augmentation to enhance model generalization.
2. **Model Development:** Design and train a robust Convolutional Neural Network (CNN) architecture optimized for emotion classification from grayscale facial images.
3. **Real-time Inference:** Implement a system capable of performing facial emotion recognition in real-time using a computer's webcam.
4. **Interactive Application:** Develop a user-friendly web interface using Streamlit to display the real-time emotion predictions and provide an intuitive user experience.
5. **Performance Optimization:** Implement strategies to minimize latency and ensure a smooth real-time experience.

II. Dataset Description (FER-2013)

A. Dataset Source & Characteristics

The FER-2013 (Facial Expression Recognition 2013) dataset was used for training and evaluating the emotion recognition model. It is a publicly available dataset consisting of grayscale images of faces, each labeled with one of seven universal emotional expressions.

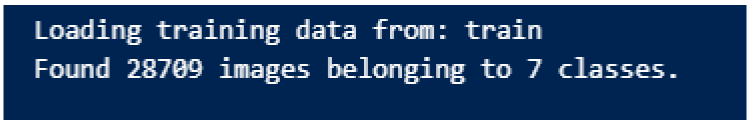
- **Source:** Publicly available (e.g., Kaggle, Challenges in Representation Learning: Facial Expression Recognition Challenge).
- **Image Format:** Grayscale images.
- **Image Resolution:** All images are 48x48 pixels.
- **Classes:** 7 distinct emotion categories:

  Angry, Disgust, Fear, Happy, Neutral, Sad, Surprise

B. Dataset Split

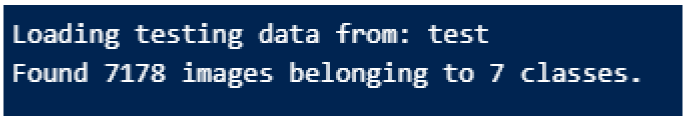
The dataset is pre-divided into training and testing sets, ensuring a standardized evaluation process.

- **Training Set:** Used to train the CNN model.



```
Loading training data from: train
Found 28709 images belonging to 7 classes.
```

- **Testing Set:** Used for model evaluation.



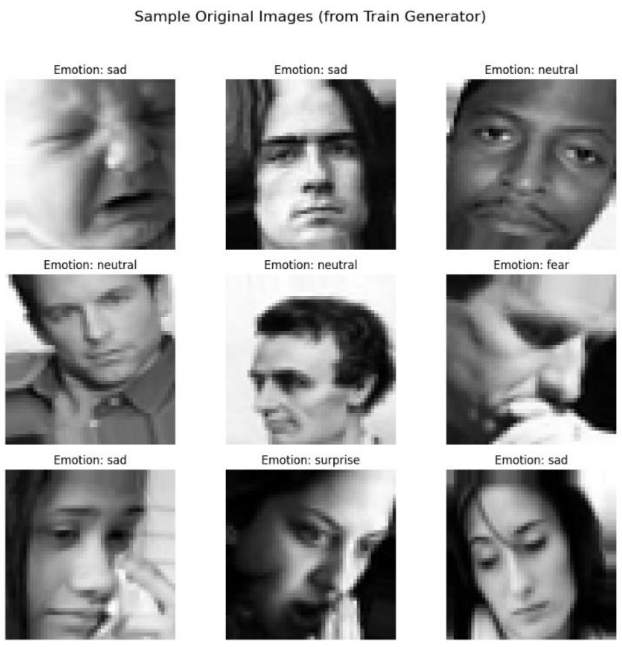
```
Loading testing data from: test
Found 7178 images belonging to 7 classes.
```

C. Data Preprocessing & Augmentation

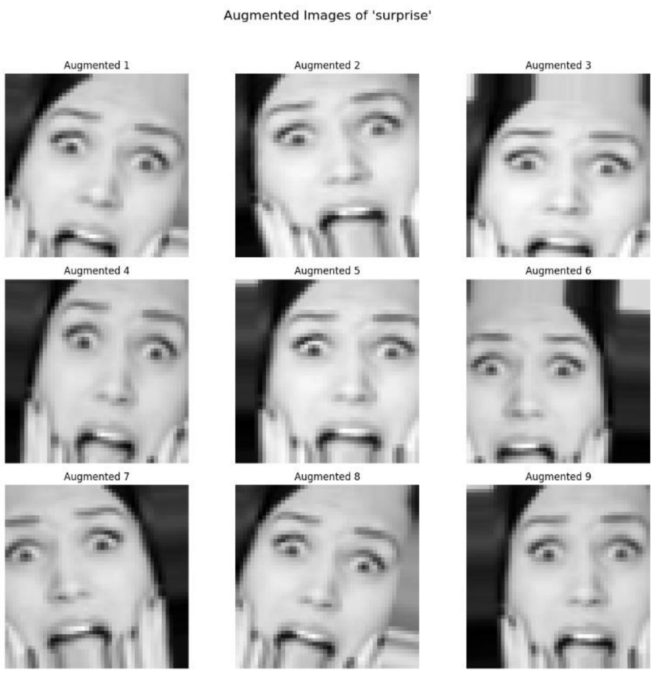
To enhance the model's ability to generalize and prevent overfitting, the following preprocessing and augmentation techniques were applied using TensorFlow's ImageDataGenerator:

- **Rescaling:** Pixel values normalized from [0, 255] to [0, 1].
- **Data Augmentation (Training Set Only):**
    - rotation_range=15
    - width_shift_range=0.1
    - height_shift_range=0.1
    - shear_range=0.1
    - zoom_range=0.1
    - horizontal_flip=True
    - fill_mode='nearest'

Sample Original Images:



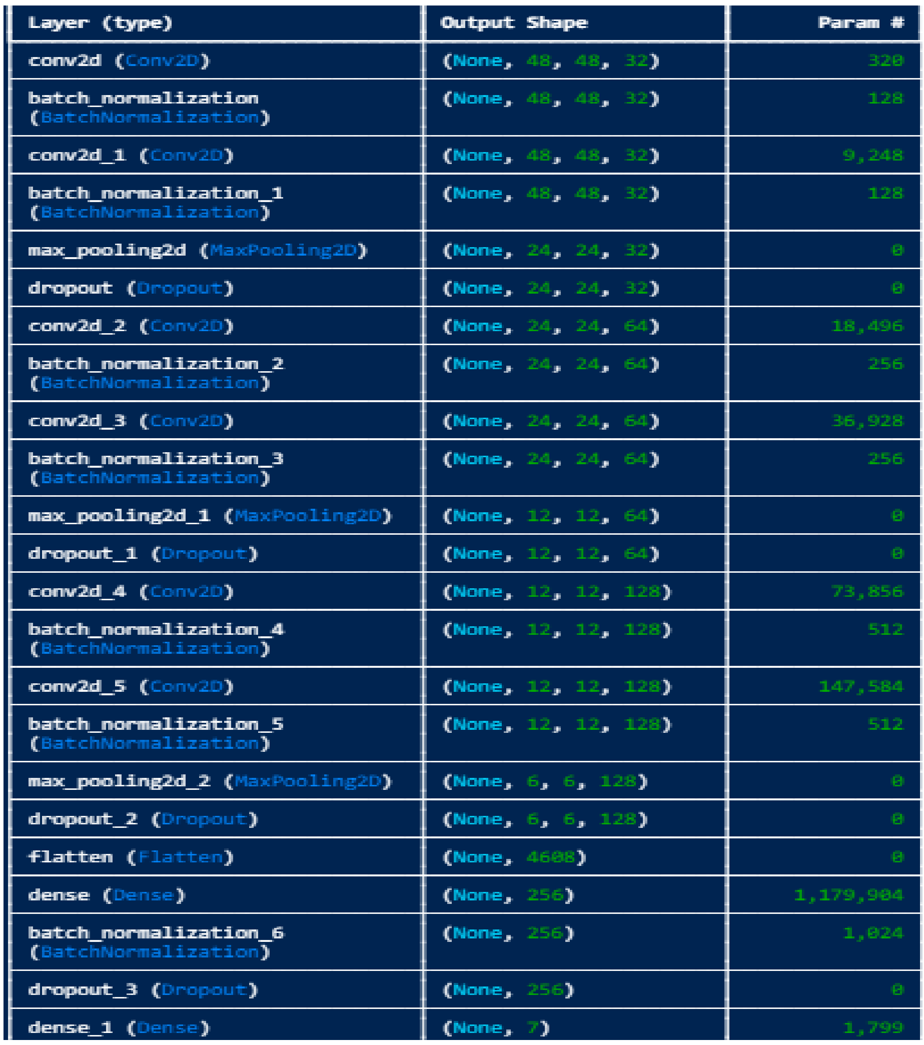
Sample Augmented Images:



III. Model Architecture Diagram

The emotion recognition model is a Convolutional Neural Network (CNN) with a VGG-like architecture, incorporating Batch Normalization and Dropout layers for improved stability and regularization. The input is a 48x48 grayscale image (1 channel).

Model Summary:



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	320
batch_normalization (BatchNormalization)	(None, 48, 48, 32)	128
conv2d_1 (Conv2D)	(None, 48, 48, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 48, 48, 32)	128
max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)	0
dropout (Dropout)	(None, 24, 24, 32)	0
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 24, 24, 64)	256
conv2d_3 (Conv2D)	(None, 24, 24, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 24, 24, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
conv2d_4 (Conv2D)	(None, 12, 12, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 12, 12, 128)	512
conv2d_5 (Conv2D)	(None, 12, 12, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_2 (Dropout)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 256)	1,179,904
batch_normalization_6 (BatchNormalization)	(None, 256)	1,024
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 7)	1,799

Architectural Breakdown:

- **Input Layer:** (48, 48, 1) grayscale images.
- **Block 1:**
  - Conv2D(32, (3,3), activation='relu', padding='same')
  - BatchNormalization()
  - Conv2D(32, (3,3), activation='relu', padding='same')

    - BatchNormalization()
    - MaxPooling2D((2,2))
    - Dropout(0.25)
- **Block 2:**
    - Conv2D(64, (3,3), activation='relu', padding='same')
    - BatchNormalization()
    - Conv2D(64, (3,3), activation='relu', padding='same')
    - BatchNormalization()
    - MaxPooling2D((2,2))
    - Dropout(0.25)
- **Block 3:**
    - Conv2D(128, (3,3), activation='relu', padding='same')
    - BatchNormalization()
    - Conv2D(128, (3,3), activation='relu', padding='same')
    - BatchNormalization()
    - MaxPooling2D((2,2))
    - Dropout(0.25)
- **Classification Head:**
    - Flatten()
    - Dense(256, activation='relu')
    - BatchNormalization()
    - Dropout(0.5)
    - Dense(7, activation='softmax') (Output layer for 7 emotion classes)

IV. Training Logs & Charts

A. Training Setup

- **Optimizer:** Adam (learning_rate=0.001)
- **Loss Function:** categorical_crossentropy
- **Metrics:** accuracy
- **Epochs:** Maximum 50
- **Batch Size:** 64

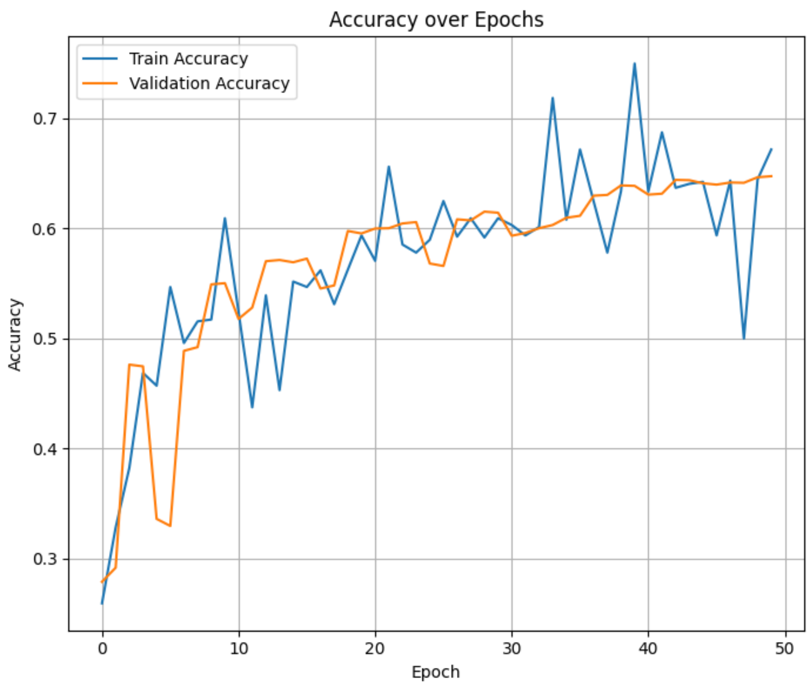
- **Callbacks:**
    - ModelCheckpoint: Saves best model based on val_accuracy (mode='max', saves to models/emotion_model_best.h5).
    - EarlyStopping: Stops if val_accuracy doesn't improve for 15 epochs (patience=15), restores best weights.
    - ReduceLROnPlateau: Reduces learning rate by factor 0.2 if val_accuracy plateaus for 7 epochs (patience=7), min_lr=0.00001.

B. Analysis of Training Log Output

- The first epoch ran for all 448 steps, taking approximately 288 seconds. The model started with low training accuracy (0.2254) and high loss, typical for an untrained state. Crucially, the validation accuracy immediately improved to 0.2789, and the model was saved by ModelCheckpoint. The HDF5 warning is a Keras recommendation for a newer saving format, which can be safely ignored for current functionality.
- The "full" epochs (e.g., Epoch 3, 5, 7, etc.) show consistent improvement. The validation accuracy generally increased, reaching significant milestones such as: 0.4763% (Epoch 3), 0.5492% (Epoch 9), 0.6059% (Epoch 24), and ultimately 0.6476% (Epoch 50).
- Around Epoch 37, the ReduceLROnPlateau callback triggered, reducing the learning rate from 0.001 to 0.0002. This usually happens when the model's validation performance plateaus, helping it find finer optima and spurring further improvements in validation accuracy.
- The training completed after its maximum allowed 50 epochs. The EarlyStopping callback did not trigger, indicating that the model continued to show (even if minor) improvements in validation accuracy until the very end. The best model weights were restored from Epoch 50, which achieved the highest validation accuracy during the run.

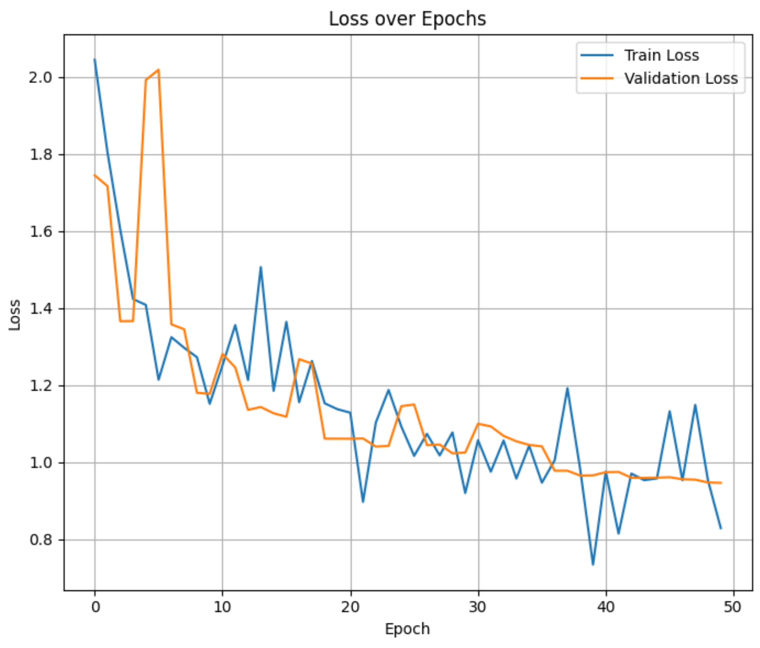
C. Training & Validation Accuracy

This plot illustrates how the model's accuracy on both the training and validation datasets evolved over each epoch.



Analysis: The training accuracy consistently increased, indicating the model was learning from the data. The validation accuracy also increased steadily, demonstrating the model's ability to generalize to unseen data. A relatively close gap between training and validation accuracy suggests good generalization and minimal overfitting, likely aided by data augmentation and dropout layers.

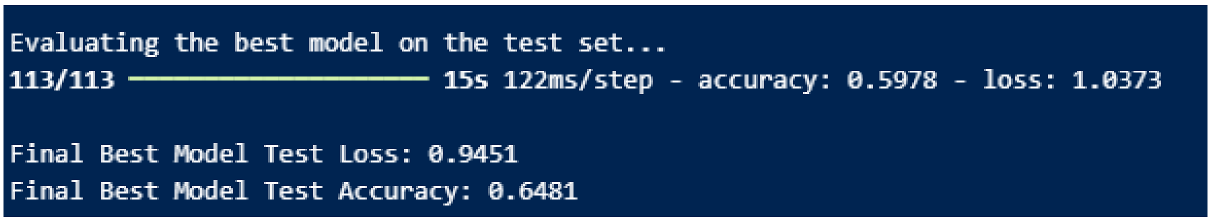
D. Training & Validation Loss



Analysis: Both training and validation loss generally decreased over epochs, confirming that the model was effectively minimizing its errors. Similar to accuracy, the validation loss largely followed the training loss, further supporting good generalization.

E. Final Model Evaluation

After training, the best model (saved by ModelCheckpoint at Epoch 50) was evaluated on the complete test set.



```
Evaluating the best model on the test set...
113/113 ━━━━━━━━━━━━━━━━━━━━ 15s 122ms/step - accuracy: 0.5978 - loss: 1.0373

Final Best Model Test Loss: 0.9451
Final Best Model Test Accuracy: 0.6481
```

V. Real-time Application & Sample Predictions

A. Technologies Used

- **Streamlit:** For creating the interactive web application interface.
- **OpenCV (cv2):** For camera access, real-time video stream processing, and face detection.
- **TensorFlow/Keras:** For loading and running inference on the trained emotion recognition model.
- **Haar Cascade Classifier:** (haarcascade_frontalface_default.xml) used for efficient initial face detection.

B. Real-time Workflow

1. The Streamlit app starts and loads the pre-trained emotion CNN model and the Haar Cascade face detector.
2. Upon clicking "Start Camera," OpenCV initiates a connection to the default webcam.
3. Frames are continuously captured. For performance optimization, only selected frames are processed.
4. Each frame is converted to grayscale. A downscaled version of the grayscale frame is used for faster face detection with the Haar Cascade classifier.
5. If faces are detected, their bounding box coordinates are scaled back to the original frame size.
6. Each detected face region-of-interest (ROI) is cropped, resized to 48x48 pixels, converted to grayscale, and normalized.
7. The preprocessed face ROI is fed into the CNN model for emotion prediction.
8. The predicted emotion (e.g., 'happy', 'sad') and its confidence score are drawn onto the original video frame along with a bounding box around the face.

9. The annotated frame is then displayed in the Streamlit interface, providing real-time visual feedback.

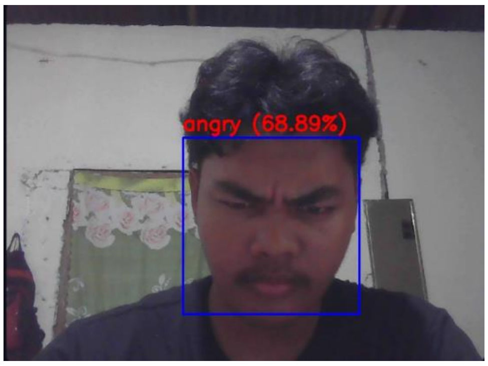
C. Performance Optimizations

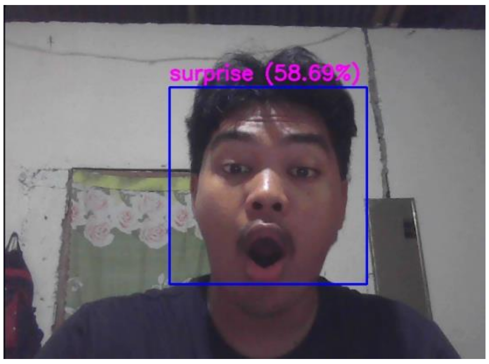
To address potential lag in real-time processing:

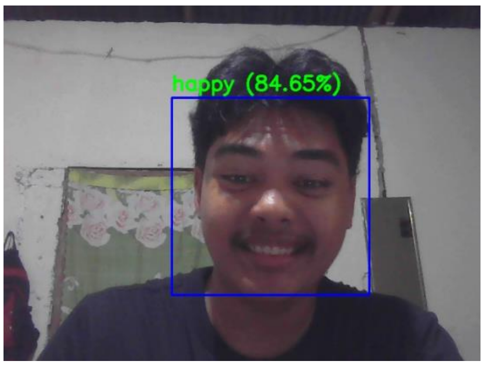
- **Frame Skipping:** Only a subset of frames (e.g., 15 FPS) are processed for emotion recognition, reducing the computational load.
- **Face Detection Downscaling:** Face detection is performed on a smaller version of the frame, as Haar Cascades are computationally less expensive on lower resolutions.
- **Cached Resources:** Model and Haar Cascade loading are cached using @st.cache_resource to ensure they are loaded only once during app startup.
- **(Optional) Camera Backend:** Explicitly using cv2.CAP_DSHOW (on Windows) to potentially improve camera performance.

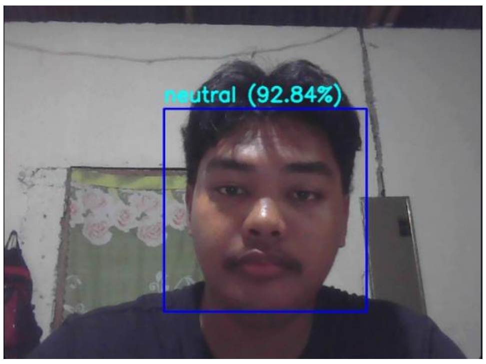
D. Sample Predictions

Below are screenshots from the live Streamlit application, demonstrating the real-time emotion recognition.









VI. Limitations & Future Improvements

A. Limitations

1. **FER-2013 Dataset Bias:** The dataset is known to be somewhat noisy, with some images mislabeled or ambiguous. Its fixed resolution (48x48) also limits the detail available.
2. **Generalization to Real-world Scenarios:** The model might struggle with extreme head poses, partial occlusions (e.g., glasses, hands over face), varying lighting conditions, or individuals not represented in the training data.
3. **Haar Cascade Detector:** While fast, Haar Cascades are prone to false positives/negatives and are less robust than modern deep learning-based face detectors, especially for faces at angles or in complex backgrounds.
4. **Categorical Emotion Model:** Emotions are complex and often continuous rather than strictly categorical. The model predicts a single discrete emotion, which is a simplification of human emotional expression. Model Performs well on emotion **anger, surprise, happy, and neutral** and perform poorly on **sad and disgust.**
5. **Performance on CPU:** Without GPU acceleration, real-time inference speed can still be limited on less powerful machines, even with optimizations.

B. Future Improvements

1. **Advanced Face Detection:** Replace Haar Cascades with more robust and accurate deep learning face detectors like MTCNN, MediaPipe Face Detection, or RetinaFace for better initial face localization.
2. **Larger/More Diverse Datasets:** Train on more comprehensive datasets (e.g., RAF-DB, AffectNet) that include a wider range of expressions, poses, and environmental conditions.
3. **Model Architecture Refinement:** Experiment with deeper or more sophisticated CNN architectures (e.g., ResNet, EfficientNet variants) or explore transfer learning from pre-trained image classification models.
4. **Model Optimization for Deployment:** Implement techniques like **model quantization** (e.g., TensorFlow Lite models) or pruning to reduce model size and inference latency, especially for deployment on edge devices or mobile.
5. **Emotion Intensity/Regression:** Instead of discrete categories, explore models that predict emotion intensity on a continuous scale (e.g., valence-arousal space).
6. **Multi-Modal Emotion Recognition:** Incorporate other cues like audio (speech tone) or body language for more accurate emotion recognition.
7. **Error Analysis:** Perform a detailed analysis of misclassified samples to identify common failure modes and target specific improvements.

VIII. References

- **Dataset:**
    - **FER-2013 Dataset:**
        - Kaggle: https://www.kaggle.com/datasets/msambare/fer2013
        - Original Challenge (Challenges in Representation Learning: Facial Expression Recognition Challenge 2013): While a direct paper is hard to find for the dataset itself, it originated from this challenge. You can refer to resources discussing the challenge or related benchmarks.
- **Libraries (Official Documentation):**
    - **TensorFlow:** https://www.tensorflow.org/
    - **Keras:** https://keras.io/
    - **OpenCV:** https://opencv.org/
    - **Streamlit:** https://streamlit.io/
    - **NumPy:** https://numpy.org/
    - **Matplotlib:** https://matplotlib.org/
- **Concepts (Key Resources for Understanding Deep Learning Fundamentals):**
    - **Convolutional Neural Networks (CNNs):**
        - Stanford CS231n Convolutional Neural Networks for Visual Recognition: https://cs231n.github.io/convolutional-networks/
        - Deep Learning Book (Goodfellow, Bengio, Courville) - Chapter 9: https://www.deeplearningbook.org/contents/convnets.html
    - **Data Augmentation:**
        - Keras ImageDataGenerator Documentation: https://keras.io/api/preprocessing/image/#imagedatagenerator-class
    - **Batch Normalization:**
        - Original Paper (Ioffe & Szegedy, 2015): https://arxiv.org/abs/1502.03167
        - Keras BatchNormalization Documentation: https://keras.io/api/layers/normalization_layers/batch_normalization/
    - **Dropout Regularization:**
        - Original Paper (Hinton et al., 2012): https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
        - Keras Dropout Documentation: https://keras.io/api/layers/regularization_layers/dropout/

- **Adam Optimizer:**
    - Original Paper (Kingma & Ba, 2014): https://arxiv.org/abs/1412.6980
    - Keras Adam Optimizer Documentation: https://keras.io/api/optimizers/adam/
- **Keras Callbacks (Model Checkpointing, Early Stopping, ReduceLROnPlateau):**
    - Keras Callbacks Guide: https://keras.io/api/callbacks/
    - ModelCheckpoint: https://keras.io/api/callbacks/model_checkpoint/
    - EarlyStopping: https://keras.io/api/callbacks/early_stopping/
    - ReduceLROnPlateau: https://keras.io/api/callbacks/reduce_lr_on_plateau/
