Real-Time Emotion Recognition

Using CNN Trained on the FER-2013 Dataset

Contents

1 Introduction **2**

2 FER-2013 Dataset **2**

3 Data Preprocessing and Augmentation **2**

3.1 Why Preprocessing? 2

3.2 Data Augmentation Techniques Used 3

4 CNN Architecture **3**

4.1 Layer-by-Layer Breakdown 4

4.2 Why CNNs Work Well for Emotion Recognition 4

5 Training Configuration **4**

5.1 Loss Function: Categorical Crossentropy 4

5.2 Optimizer: Adam 5

5.3 Callbacks Used 5

6 Training Analysis **6**

6.1 Final Training Metrics 6

6.2 Graphs and Observations 6

6.3 Understanding the Graphs 7

7 Saving the Model **7**

8 Real-Time Emotion Detection **7**

8.1 Overview of `main.py` 7

8.2 Haar Cascade Classifier: `haarcascade_frontalface_default.xml` 8

8.3 Code Flow 9

9 Conclusion **10**

10 Future Work **10**

1 Introduction

Facial expression is one of the most powerful, natural, and universal forms of human communication. This project aims to classify human emotions in real-time using facial images, leveraging the power of Convolutional Neural Networks (CNNs) trained on the FER-2013 dataset.

The model is saved as `best_model.keras` and deployed through `main.py`, which accesses webcam input, performs face detection, and displays predicted emotion labels live on screen.

2 FER-2013 Dataset

The FER-2013 dataset is a publicly available benchmark dataset released in the ICML 2013 Challenge. It contains 35,887 grayscale images of faces (48x48 pixels) labeled into seven categories:
Emotion Classes: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral

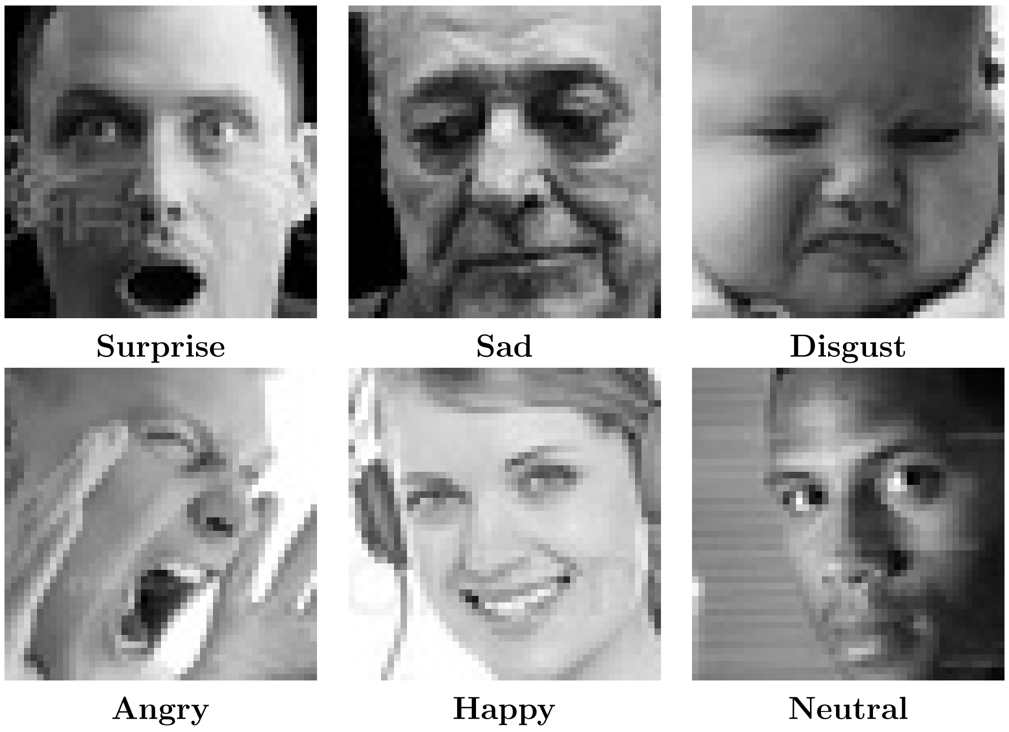


Figure 1: Sample images from FER-2013 dataset for six emotion classes

3 Data Preprocessing and Augmentation

3.1 Why Preprocessing?

Raw images are not suitable for direct input into neural networks. Hence, preprocessing ensures:

- Pixel normalization (scaling from [0, 255] to [0, 1])
- Dimensionality consistency (all inputs are 48x48)
- Balanced generalization via augmentation

3.2 Data Augmentation Techniques Used

To combat overfitting due to limited dataset size, augmentation artificially increases data variety. The following transformations were applied:

- **Rotation Range:** Randomly rotate images within $\pm 30^\circ$
- **Zoom Range:** Random zoom (up to 20%)
- **Width/Height Shift:** Shift images horizontally/vertically
- **Horizontal Flip:** Mirror the image to simulate side symmetry

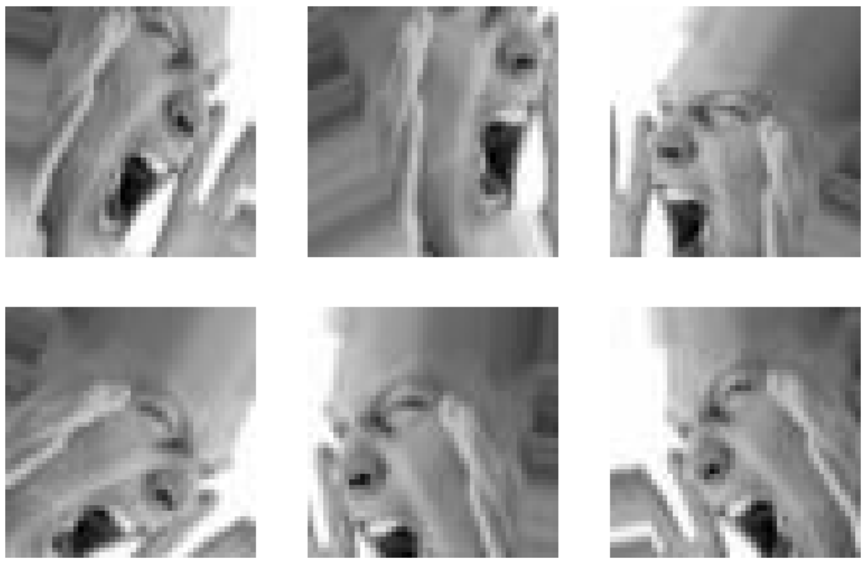


Figure 2: Impact of data augmentation on a sample image

4 CNN Architecture

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed to work with image data. They exploit spatial hierarchies through local connectivity and shared weights.

4.1 Layer-by-Layer Breakdown

The model consists of:

- **Convolutional Layers:** Apply filters to extract low- to high-level features.
- **ReLU Activation:** Introduce non-linearity.
- **Batch Normalization:** Stabilize and accelerate training.
- **MaxPooling:** Downsample the spatial dimensions.
- **Dropout:** Randomly deactivate neurons to prevent overfitting.
- **Fully Connected (Dense) Layers:** Perform classification.

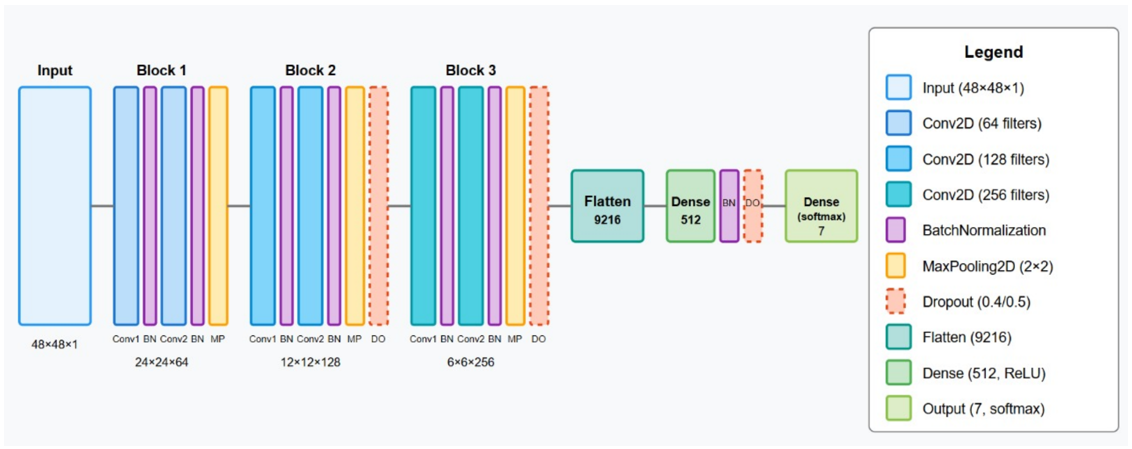


Figure 3: CNN model used for FER-2013 classification

4.2 Why CNNs Work Well for Emotion Recognition

CNNs learn hierarchical features:

- Lower layers detect edges and blobs.
- Middle layers learn parts of facial features (e.g., eyes, mouth).
- Upper layers assemble these into emotion-relevant patterns.

5 Training Configuration

5.1 Loss Function: Categorical Crossentropy

Since the problem involves multi-class classification, categorical crossentropy is used:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where y_i is the true label and $\hat{y}_i$ is the predicted probability.

5.2 Optimizer: Adam

Adam (Adaptive Moment Estimation) is used for optimization. It combines momentum and RMSProp, adjusting learning rates based on both the mean and variance of gradients.

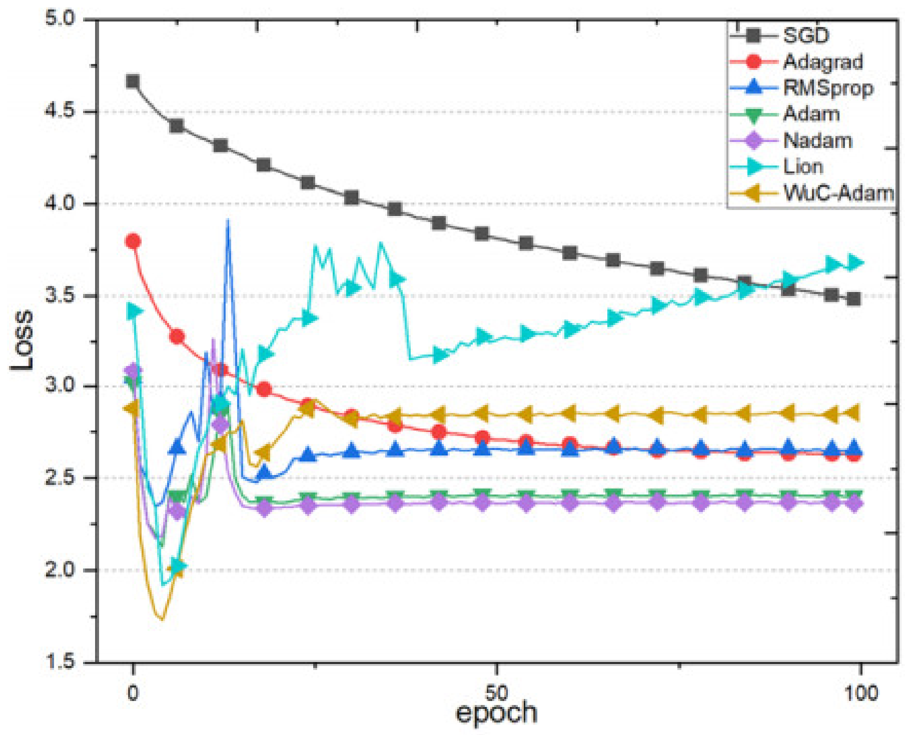


Figure 4: Visual comparison of optimizers

5.3 Callbacks Used

- **ModelCheckpoint:** Saves the best-performing model during training.
- **EarlyStopping:** Prevents overfitting by halting when validation loss plateaus.
- **ReduceLROnPlateau:** Decreases learning rate when model hits a plateau.

6 Training Analysis

6.1 Final Training Metrics

The model was trained for **50** epochs, and the best performing model (saved via `ModelCheckpoint`) achieved the following results:

- **Final Training Accuracy:** 66.32 %
- **Final Validation Accuracy:** 67.20 %
- **Final Training Loss:** 1.1435
- **Final Validation Loss:** 1.1480

These metrics suggest that the model generalized reasonably well on unseen data. Validation performance closely tracked training performance, indicating little to no overfitting.

6.2 Graphs and Observations

Training and validation accuracy and loss were plotted across epochs to monitor model behavior and detect overfitting, underfitting, or convergence issues.

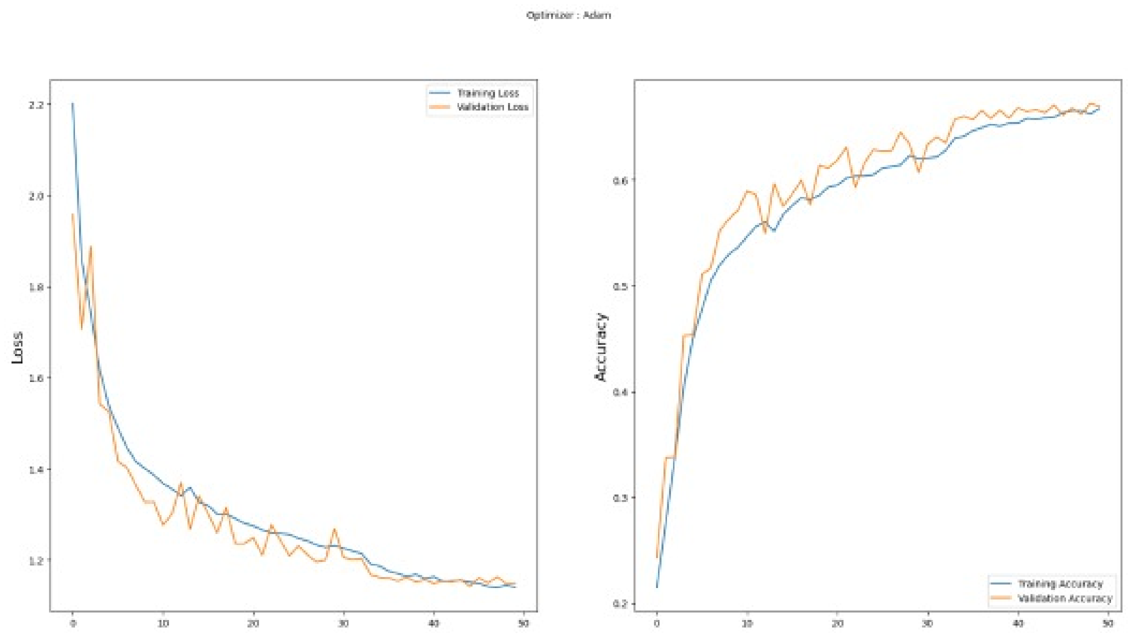


Figure 5: Accuracy and Loss vs Epochs during training

6.3 Understanding the Graphs

- **Accuracy Peaks:** Sudden increases in training or validation accuracy usually indicate that the model has learned a new generalization pattern.
- **Loss Valleys:** These signify reductions in the model's error. Deep valleys in validation loss usually mean that the model is performing well on unseen data.
- **Validation Divergence:** If validation loss increases while training loss continues to drop, the model may be overfitting.
- **Convergence:** A flattening of both accuracy and loss curves indicates convergence to an optimal solution.

Interpretation: From the plots, it is evident that the model steadily learned meaningful features over time. Both training and validation accuracy increased consistently throughout the 50 epochs, starting from around 22% and reaching over 68%, with minimal divergence between the two. This close alignment indicates that the model generalized well and did not suffer from significant overfitting. Similarly, the training and validation loss curves decreased monotonically from over 2.2 to about 1.15, suggesting continuous improvement in minimizing error. Notably, the validation loss remained slightly below the training loss for most of training, which can occur due to dropout regularization or batch normalization. Overall, the trends reflect healthy convergence behavior, with the Adam optimizer facilitating stable and efficient learning throughout.
Note: Interestingly, the validation accuracy remains slightly higher than the training accuracy throughout most of the training. This behavior is likely due to the use of regularization techniques like dropout, which are only active during training, making it noisier. During validation, the model operates in inference mode without dropout, allowing it to perform more consistently. This is a common and acceptable occurrence in well-regularized models.

7 Saving the Model

The final model is saved using:

```
model.save("best_model.keras")
```

This ensures future use without retraining.

8 Real-Time Emotion Detection

8.1 Overview of `main.py`

The script `main.py` serves as the deployment interface for real-time emotion recognition. It utilizes a webcam feed and processes each frame as follows:

1. OpenCV captures a frame from the webcam.

2. The frame is converted to grayscale for faster processing.
3. Haar Cascade is used to detect face bounding boxes.
4. Each detected face (Region of Interest, ROI) is:
    - Cropped and resized to 48x48
    - Normalized (pixel values scaled to [0, 1])
    - Converted into a NumPy array and reshaped for model input
5. The processed ROI is passed through the CNN model.
6. The emotion label is predicted and rendered onto the video frame.

8.2 Haar Cascade Classifier: `haarcascade_frontalface_default.xml`

Haar Cascades are object detection algorithms introduced by Viola and Jones. OpenCV provides pre-trained classifiers like `haarcascade_frontalface_default.xml` for face detection. Here's how it works:

- **Haar-like Features:** These are binary patterns used to capture differences in intensity (e.g., bridge of nose vs eyes).
- **Integral Images:** Used to speed up computation of Haar features in constant time.
- **Cascade Classifier:** A series of stages, each of which must be passed for a window to be declared as a face. Early stages eliminate non-faces quickly.
- **Sliding Window:** The classifier scans the entire image at multiple scales.

Haar Cascades are fast, lightweight, and ideal for real-time applications despite being less accurate than deep learning-based detectors.

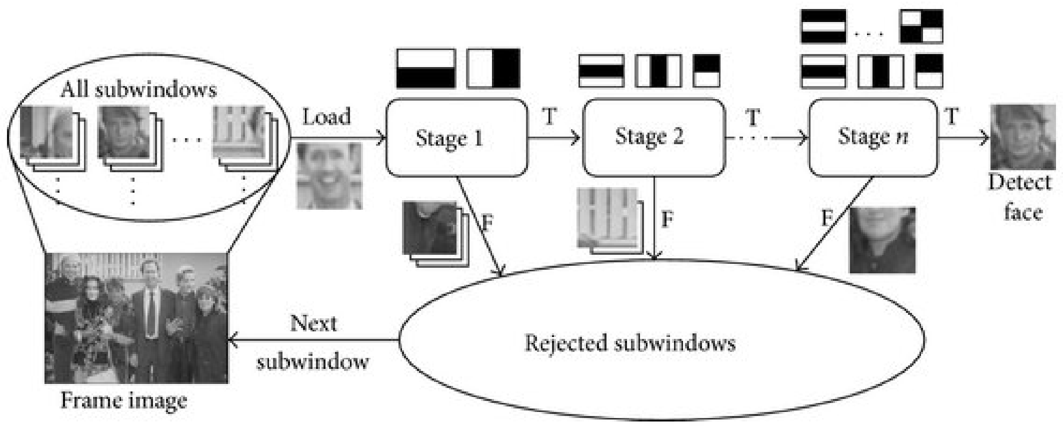


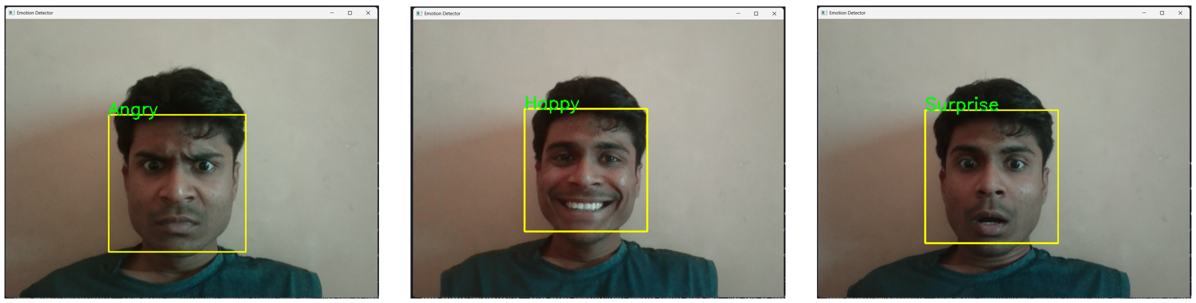
Figure 6: Haar Cascade face detection pipeline

8.3 Code Flow

The core flow in the `main.py` script can be outlined as:

```
# Load Haar Cascade and CNN Model
face_classifier = cv2.CascadeClassifier('
   haarcascade_frontalface_default.xml')
classifier = load_model('best_model.keras')

# Start video capture
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    frame = cv2.flip(frame, 1)  # Mirror the frame
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = face_classifier.detectMultiScale(gray)

    for (x, y, w, h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        roi_gray = cv2.resize(roi_gray, (48, 48))
        roi = roi_gray.astype("float") / 255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi, axis=0)

        prediction = classifier.predict(roi, verbose=0)[0]
        label = emotion_labels[prediction.argmax()]
        cv2.putText(frame, label, (x, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        cv2.rectangle(frame, (x, y), (x+w, y+h),
                      (0, 255, 255), 2)

    cv2.imshow('Emotion␣Detector', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Listing 1: Real-time Emotion Detection Pipeline



Angry **Happy** **Surprise**

Figure 7: Real-time webcam feed with emotion classification overlay

9 Conclusion

This project demonstrates an end-to-end deep learning pipeline for real-time facial emotion recognition using CNNs. From training on FER-2013 to deploying with OpenCV, it validates the real-world applicability of deep learning in affective computing.

10 Future Work

- Use face alignment techniques before prediction.
- Experiment with deeper architectures (e.g., ResNet).
- Deploy to mobile using TensorFlow Lite.
- Add temporal smoothing to stabilize predictions.