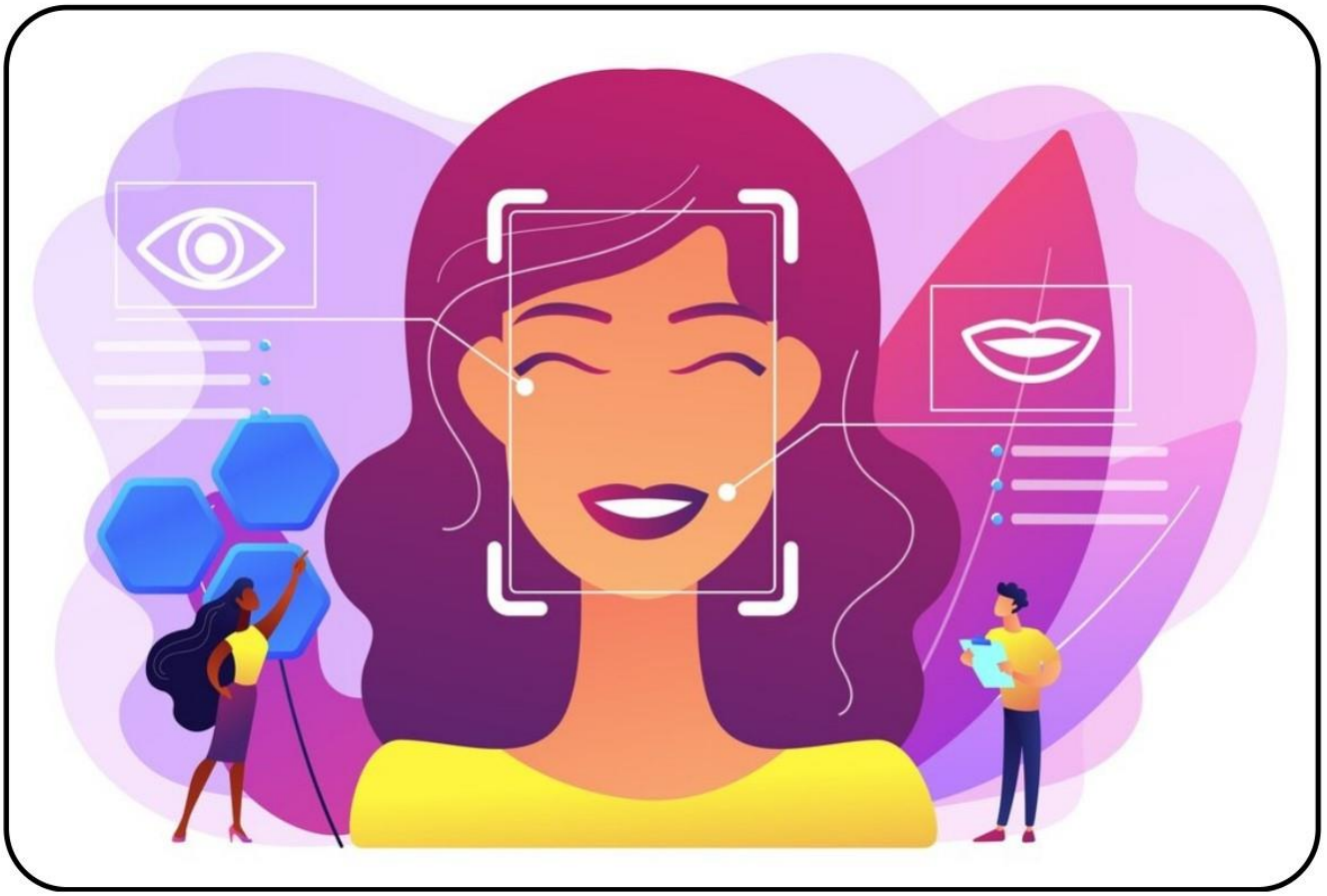


EmotiQuotient

Identification of Human Emotions

ABSTRACT

Emotion detection in humans is a crucial aspect of human-computer interaction, with applications ranging from mental health monitoring to enhancing user experience in interactive systems. This study presents a deep learning-based approach for identifying human emotions from facial expressions. The proposed method leverages a convolutional neural network (CNN) architecture trained on a large dataset of labelled facial images to classify emotions into categories such as anger, disgust, fear, happiness, neutrality, sadness, and surprise.

The model is built using Keras with TensorFlow as the backend, incorporating state-of-the-art techniques in image processing and facial feature extraction. The system first detects faces in the input images using a pre-trained Haar Cascade classifier, followed by emotion classification using the trained CNN model. The model's accuracy is evaluated using a test dataset, and its performance is compared with existing emotion detectionframeworks.

Our results demonstrate that the proposed method achieves high accuracy in real-time emotion detection tasks, making it suitable for integration into applications such as emotion-aware assistive technologies, interactive games, and customer service systems. The study also discusses potential challenges and future directions, including improving model robustness across diverse demographic groups and enhancing the model's ability to interpret subtle emotional cues.

This research contributes to the growing field of affective computing by providing an efficient and reliable tool for understanding and responding to human emotions, paving the way for more empathetic and intelligent systems.

TABLE OF CONTENTS

INDEX

Chapter 1: Introduction

1.1 Problem Statement:

To implement a machine learning model for identifying human emotions based on provided facial expressions.The goal of this project is to design and implement a deep learning-based model capable of detecting and classifying emotions from facial images. The solution must be robust, able to process real-time inputs, and adaptable to different environmental conditions. The ultimate aim is to create a reliable tool that can be integrated into various applications, enhancing the ability of machines to understand and respond to human emotions effectively.

1.2 Objectives:

Emotion detection in humans is a crucial aspect of human-computer interaction, with applications ranging from mental health monitoring to enhancing user experience in interactive systems. This study presents a deep learning-based approach to identifying human emotions from facial expressions. The proposed method leverages a convolutional neural network (CNN) architecture trained on a large dataset of labeled facial images to classify emotions such as anger, disgust, fear, happiness, neutrality, sadness, and surprise.

The model uses Keras with TensorFlow as the backend, incorporating state-of-the-art techniques in image processing and facial feature extraction. The system first detects faces in the input images using a pre-trained Haar Cascade classifier and emotion classification using the trained CNN model. The model's accuracy is evaluated using a test dataset, and its performance is compared with existing emotion detection frameworks.

Our results demonstrate that the proposed method achieves high accuracy in real-time emotion detection tasks, making it suitable for integration into applications such as emotion-aware assistive technologies, interactive games, and customer service systems This research contributes to the growing field of affective computing by providing an efficient and reliable tool for understanding and responding to human emotions, paving the way for more empathetic and intelligent systems.

1.3 Motivation:

Enhanced Human-Computer Interaction:

Emotion detection helps make interactions with computers more intuitive by recognizing and responding to human emotions. This can make virtual assistants, customer service bots, and other AI-driven interfaces more empathetic and effective.

Mental Health Monitoring:

By identifying emotional states, emotion detection can be used to monitor psychological well-being, detect early signs of mental health issues like depression or anxiety, and provide interventions in a timely manner.

Education and Learning:

In educational settings, emotion detection can help identify when a student is confused, bored, or frustrated, allowing educators or learning platforms to adjust the content or provide assistance.

Customer Service:

Businesses can use emotion recognition in customer service interactions to analyze customer satisfaction and adjust strategies to improve engagement and resolve issues.

1.4 Scope:

A facial expression can be said to be the movement of muscles beneath the skin of the face. Facial expressions are a form of nonverbal communication. The human face can convey countless emotions without saying a single word. Some forms of nonverbal communication are not universal but these facial expressions are universal and can be understood by any kind of people. The facial expressions of happiness, sadness, anger, surprise, fear, and disgust are the same across people of different cultures.

The movements of muscles convey the emotions of individuals to people who see them. They are the means through which social information conveys the emotional state of an individual to observers.

Key points:

- Muscle movement:

  Facial expressions are created by the contraction of muscles beneath the facial skin.

- Nonverbal communication:

  Facial expressions are a primary form of nonverbal communication, meaning they convey emotions without spoken words.

- Universality:

  Many basic facial expressions, like those for happiness, sadness, and anger, are considered universal and understood by people across different cultures.

- Social information:

  Facial expressions are a crucial way to convey social information and understand another person's emotional state.

Chapter 2: System Requirements

2.2 SOFTWARE REQUIREMENTS

Refer Table 2.1.1

SOFTWARE	**DESCRIPTION**
Visual Studio Code	Platform for Project
MS. Office	Documentation tools
Google Chrome	Web Browser

Table 2.1.1

This table 2.1.1 gives information about software requirements

2.2 HARDWARE REQUIREMENTS

Refer Table 2.2.1

HARDWARE	**DESCRIPTION**
LAPTOP WINDOWS 10	Workspace for the Project
Hard disk space	To Save data

Table 2.2.1

This table 2.2.1 gives information about hardware requirements.

Chapter 3: Tools Used

3.1 Development Tools:

VS Code: A powerful and flexible code editor used for writing and debugging Python scripts. It supports multiple extensions for easier coding and provides debugging capabilities.

3.2 Libraries and Frameworks:

Matplotlib, Seaborn:

- Used for visualizing data and plotting graphs.
- Matplotlib helps in plotting figures, while Seaborn enhances the aesthetics of the plots.

NumPy, Pandas:

- NumPy is used for efficient numerical computations, particularly for handling arrays and matrices, which are common in image processing.
- Pandas assists in handling and analyzing structured data, though not heavily used in image processing tasks here.

OpenCV:

- Open Source Computer Vision Library, which provides tools for real-time computer vision.
- Used for detecting faces in images through CascadeClassifier, converting images to grayscale, and drawing rectangles around detected faces.

TensorFlow and Keras:

- These deep learning libraries were used to create and train the Convolutional Neural Network (CNN) for emotion detection.
- TensorFlow/Keras layers like Conv2D, MaxPooling2D, Dropout, and Dense were used to build the CNN architecture.
- ImageDataGenerator was utilized to generate batches of tensor image data with real-time data augmentation.

Callbacks in Keras:

- ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau were utilized to improve model training by saving the best model, stopping early if no improvements are seen, and adjusting the learning rate dynamically.

Haar Cascade Classifier:

- A pre-trained classifier (haarcascade_frontalface_default.xml) for face detection. It quickly identifies faces in an image, and this is followed by emotion classification.

Image Processing Tools:

- Functions like img_to_array, cv2.imread, cv2.cvtColor, cv2.resize, and cv2.putText handle image reading, conversion, resizing, and labeling with emotion predictions.

3.3 Model Building Tools

- **Convolutional Neural Network (CNN)**:
  - A sequential model was built with multiple layers of convolution, batch normalization, and pooling to capture spatial hierarchies in images.
  - The model uses categorical cross-entropy loss and the Adam optimizer for classification across 7 different emotions.

3.4 Data Augmentation Tools

- **ImageDataGenerator:**

ImageDataGenerator from Keras is used to perform real-time data augmentation. This helps improve model generalization by applying transformations like rescaling, rotation, zooming, and horizontal flipping to the input images.

  - Train Set Augmentation: It augments the training set to create diverse variations of the data.
  - Validation Set Handling: The validation set is processed similarly, but without any augmentation, to maintain evaluation consistency.

3.5 Processing Tools

- **Emotion Classification**:

  - After detecting faces using the Haar Cascade classifier, the model processes each detected face to predict its emotion. The emotion is then displayed on the image using OpenCV's cv2.putText function.
  - Predicted emotions are categorized into 7 distinct labels: Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise.

Chapter 4: System Design

4.1 System Architecture

Human behavior is deeply influenced by emotions. Detection of emotions plays a pivotal role in understanding how individuals respond to various stimuli, such as reading text, encompassing feelings of anger, anxiety, confusion, or nervousness.

The proposed human emotion recognition system is of five components: input speech signal, pre-processing, feature extraction and selection, classification and finally emotions recognition. The emotion recognition result is strongly depending on the emotional features and which provided to various types of classifier that have been used to represent the emotion

Facial emotion recognition FER typically has six steps. The first the user provides an input image, which can be uploaded from the local file system and the next step is to The system validates the uploaded image by cross-referencing it with known datasets. The third step, the system detects faces within the image using techniques like Haar Cascade classifiers. A rectangle is drawn around the detected face to isolate it for further processing. The following step is to use a key facial landmarks, such as eyes, mouth, and nose, which are detected within the face region. The system extracts spatial and temporal features from these landmarks, which are crucial for emotion analysis. In the fifth step,The extracted features are fed into a trained emotion classification model, which predicts the emotion category (e.g., Happy, Surprise, Angry, etc.). and the final step is output where predicted emotion is displayed.

4.2 Block Diagram

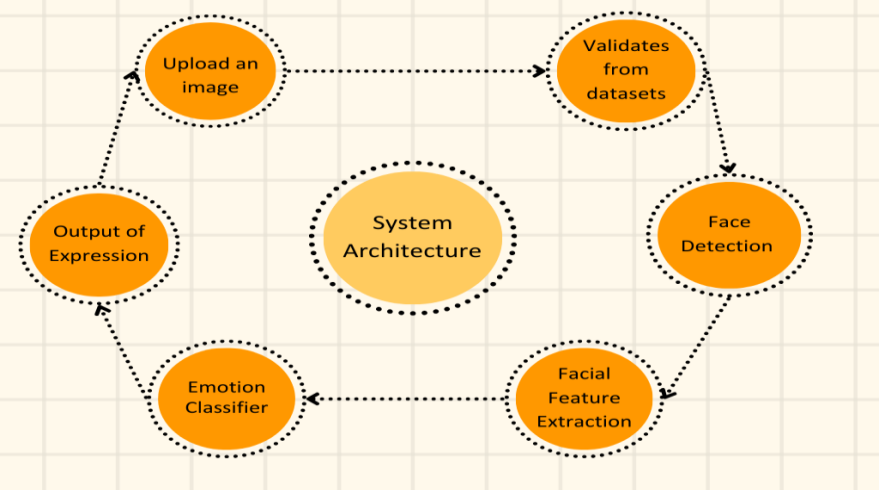


Figure 1.1

The Figure 1.1 shows Block Diagram of Emotion Detection Identification

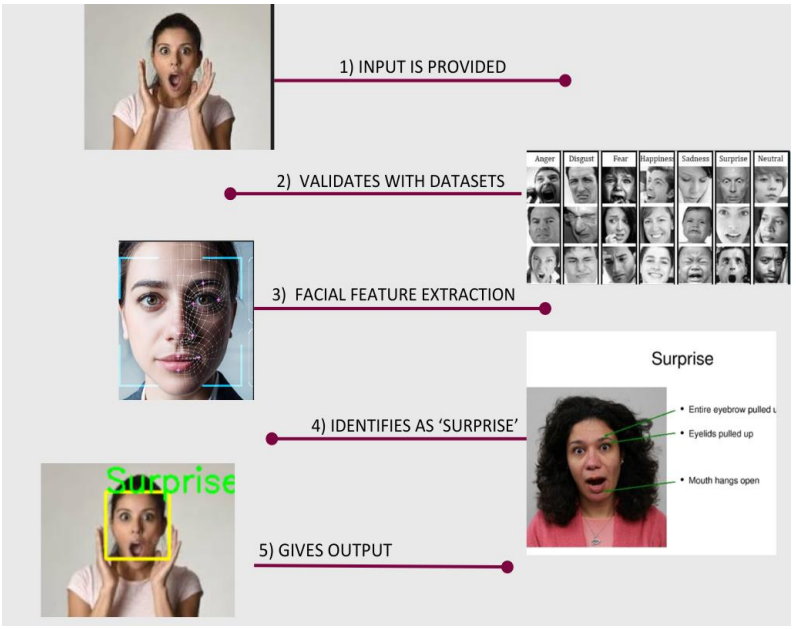


Figure 1.2

The Figure 1.2 shows the procedure for an input image where a face region and facial landmarks are detected.

Figure 1.3

Figure 1.3 shows that Facial landmarks are visually salient points such as the end of a nose, and the ends of eyebrows and the mouth. The pairwise positions of two landmark points or the local texture of a landmark are used as features.

Chapter 5: Implementation

5.1 Implementation:

The implementation of the Facial Emotion Recognition (FER) system involves several steps to create an end-to-end pipeline for emotion detection using deep learning and computer vision. The following outlines the key aspects of the implementation process.

Dataset Preparation:

- The datasets are divided into two directories: train and validation. Each directory contains subfolders for different emotions (e.g., Angry, Happy, Neutral, etc.).
- The image data is preprocessed using the ImageDataGenerator class, which normalizes the images and generates batches for model training.

Face Detection and Preprocessing:

- Haar Cascade Classifier (haarcascade_frontalface_default.xml) is employed to detect faces in the input images.
- Detected faces are converted to grayscale, resized to 48x48 pixels, and normalized to a range of [0, 1] to fit the model's input requirements.

Model Training:

- A Convolutional Neural Network (CNN) is built using Keras with multiple layers of convolution, activation, pooling, and dropout. The model is compiled with categorical_crossentropy loss and trained on the preprocessed dataset.
- Training includes hyperparameter tuning, the use of checkpoints for saving the best model, and early stopping to avoid overfitting.

Model Testing:

- Once trained, the model is evaluated on the validation dataset. The final accuracy and loss are used to measure the performance of the emotion classifier.

Emotion Classification:

- After detecting a face from a new input image, the CNN model predicts the emotion by classifying it into one of the predefined categories (Angry, Happy, Neutral, etc.).
- The output is displayed by annotating the image with the predicted emotion and a bounding box around the face.

5.2 Environmental Setup

The project consists of several key modules, each responsible for a specific function in the emotion recognition. These modules work together to complete the facial emotion detection task.

1. **Face Detection Module**: Detects human faces in an input image using the Haar Cascade classifier.
    - **Input**: Raw images from the user or test dataset.
    - **Process**: The Haar Cascade classifier scans the image and detects face regions by drawing a rectangle around them.
    - **Output**: Coordinates of the detected face, which are used to crop the face region for further processing.

2. **Image Preprocessing Module**: Preprocesses detected faces by converting them to grayscale, resizing, and normalizing.
    - **Input**: Cropped face image from the face detection module.
    - **Process**: Resizes the image to 48x48 pixels and normalizes pixel values to [0, 1]. The image is reshaped to match the input format of the CNN model.
    - **Output**: Preprocessed face image ready for emotion classification

3. **CNN Model Module**: The core of the system, this module contains the Convolutional Neural Network (CNN) model that performs emotion classification.
    - **Input**: Preprocessed face image.
    - **Process**: The CNN processes the image through several layers (convolution, pooling, activation) to extract features. The fully connected layers classify the image into one of the seven emotion categories.
    - **Output**: A prediction vector that contains the probability scores for each emotion.

4. **Emotion Classification Module**: Classifies the input image into a specific emotion category.
    - **Input**: Probability scores from the CNN model.
    - **Process**: The highest probability score is selected, and the corresponding emotion label is returned.
    - **Output**: The predicted emotion (e.g., Happy, Sad, Surprise).

5. **Result Display Module**: Displays the annotated image with a bounding box around the face and the predicted emotion.
    - **Input**: The original image and the predicted emotion label.
    - **Process**: Using OpenCV's putText function, the predicted emotion is written on the image near the face.
    - **Output**: The final output image with emotion labels displayed on-screen.

Chapter 6: Results

6.1 Steps for Output

Step-1: Open python file.

Step-2: Provide the correct image path in ‘image_path’.

Figure 6.1 shows image path given

```
image_path = r'C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\.ipynb_checkpoints\images\e.jpeg'
```

Figure 6.1

Step-3: Run the code

```
2024-09-14 18:14:50.419918: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different
omputation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-09-14 18:14:56.937205: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different
omputation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\BUSIREDDY ADITI\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
ayer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-09-14 18:15:15.148312: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
C:\Users\BUSIREDDY ADITI\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\optimizers\base_optimizer.py:33: UserWarning: Argument `decay` is no longer supported and will be ignored.
  warnings.warn(
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Error in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimizer.
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 650ms/step
```

Figure 6.2

The Figure 6.2 shows that the terminal is successfully executed

Step-4: Output

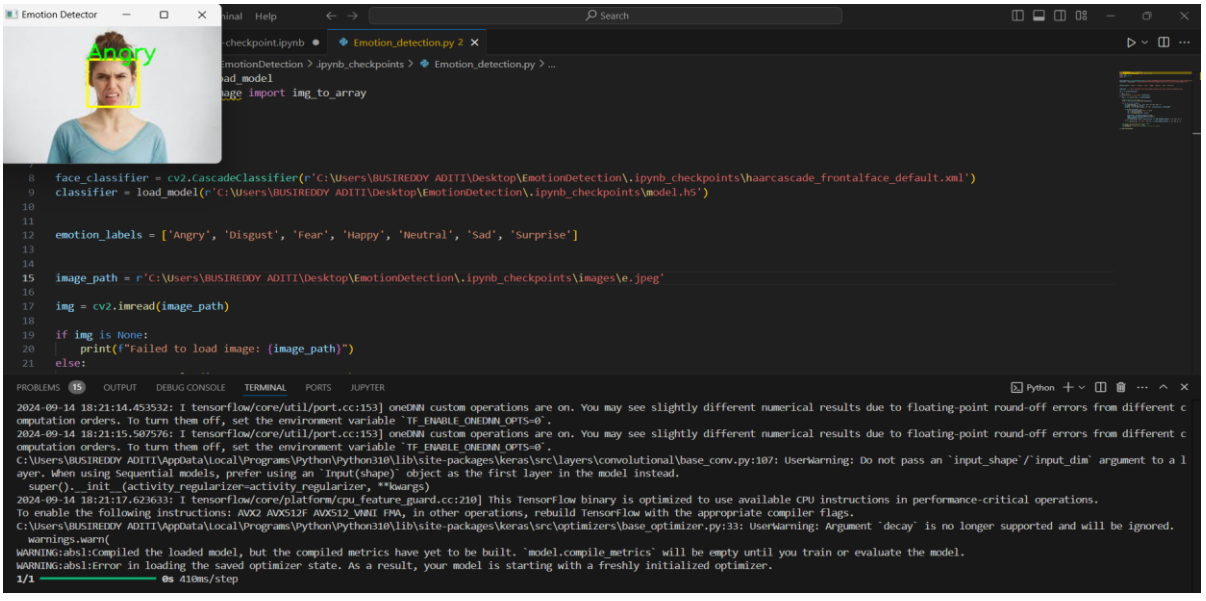


Figure 6.2

The Figure 6.2 shows the output of the given image as input.

Analysis: Analysis of outputs from emotion classification which imports datasets, trains and validates the data:-

Training and Validation Data:
Output:

```
Training directory: C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\train\images\images\train
Validation directory: C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\train\images\images\validation
Found 28821 images belonging to 7 classes.
Found 7066 images belonging to 7 classes.
```

Figure 6.3

Model Building :
Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 64)	640
batch_normalization (BatchNormalization)	(None, 48, 48, 64)	256
activation (Activation)	(None, 48, 48, 64)	0
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0
dropout (Dropout)	(None, 24, 24, 64)	0
conv2d_1 (Conv2D)	(None, 24, 24, 128)	204,928
batch_normalization_1 (BatchNormalization)	(None, 24, 24, 128)	512
activation_1 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 128)	0
dropout_1 (Dropout)	(None, 12, 12, 128)	0
conv2d_2 (Conv2D)	(None, 12, 12, 512)	590,336
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 512)	2,048
activation_2 (Activation)	(None, 12, 12, 512)	0

C: > Users > BUSIREDDY ADITI > Desktop > EmotionDetection > .ipynb_checkpoints > emotion-classification-cnn-using

Code | Markdown | Run All | Clear All Outputs | Outline ...

activation_3 (Activation)	(None, 6, 6, 512)	0
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 512)	0
dropout_3 (Dropout)	(None, 3, 3, 512)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 256)	1,179,904
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
activation_4 (Activation)	(None, 256)	0
dropout_4 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131,584
batch_normalization_5 (BatchNormalization)	(None, 512)	2,048
activation_5 (Activation)	(None, 512)	0
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 7)	3,591

Total params: 4,478,727 (17.08 MB)

Trainable params: 4,474,759 (17.07 MB)

Non-trainable params: 3,968 (15.50 KB)

Fitting the Model with Training and Validation Data :

Output:

```
Epoch 1/48
c:\Users\BUSIREDDY ADITI\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:122: UserWarning: Your `PyDa
  self._warn_if_super_not_called()
225/225 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.2576 - loss: 1.9349
Epoch 1: val_accuracy improved from -inf to 0.33210, saving model to ./model.keras
225/225 ━━━━━━━━━━━━━━━━━━━━ 429s 2s/step - accuracy: 0.2579 - loss: 1.9342 - val_accuracy: 0.3321 - val_loss: 1.9367 - learning_rate: 0.0010
Epoch 2/48
  1/225 ━━━━━━━━━━━━━━━━━━━━ 4:45 1s/step - accuracy: 0.4219 - loss: 1.4601
Epoch 2: val_accuracy did not improve from 0.33210
c:\Users\BUSIREDDY ADITI\AppData\Local\Programs\Python\Python310\lib\contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure that
  self.gen.throw(typ, value, traceback)
225/225 ━━━━━━━━━━━━━━━━━━━━ 1s 942us/step - accuracy: 0.4219 - loss: 1.4601 - val_accuracy: 0.2308 - val_loss: 2.7118 - learning_rate: 0.0010
Epoch 3/48
225/225 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.4372 - loss: 1.4631
Epoch 3: val_accuracy improved from 0.33210 to 0.42869, saving model to ./model.keras
225/225 ━━━━━━━━━━━━━━━━━━━━ 303s 1s/step - accuracy: 0.4372 - loss: 1.4629 - val_accuracy: 0.4287 - val_loss: 1.5569 - learning_rate: 0.0010
Epoch 4/48
  1/225 ━━━━━━━━━━━━━━━━━━━━ 4:28 1s/step - accuracy: 0.4219 - loss: 1.4061
Epoch 4: val_accuracy did not improve from 0.42869
225/225 ━━━━━━━━━━━━━━━━━━━━ 1s 403us/step - accuracy: 0.4219 - loss: 1.4061 - val_accuracy: 0.3462 - val_loss: 1.8981 - learning_rate: 0.0010
Epoch 5/48
225/225 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.5107 - loss: 1.2783
Epoch 5: val_accuracy improved from 0.42869 to 0.47230, saving model to ./model.keras
225/225 ━━━━━━━━━━━━━━━━━━━━ 307s 1s/step - accuracy: 0.5107 - loss: 1.2782 - val_accuracy: 0.4723 - val_loss: 1.3299 - learning_rate: 0.0010
Epoch 6/48
  1/225 ━━━━━━━━━━━━━━━━━━━━ 4:29 1s/step - accuracy: 0.5312 - loss: 1.1546
Epoch 6: val_accuracy improved from 0.47230 to 0.76923, saving model to ./model.keras
225/225 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5312 - loss: 1.1546 - val_accuracy: 0.7692 - val_loss: 1.0354 - learning_rate: 0.0010
Epoch 7/48
225/225 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.5507 - loss: 1.1813
Epoch 7: val_accuracy did not improve from 0.76923
225/225 ━━━━━━━━━━━━━━━━━━━━ 574s 3s/step - accuracy: 0.5507 - loss: 1.1813 - val_accuracy: 0.5091 - val_loss: 1.2525 - learning_rate: 0.0010
Epoch 8/48
  1/225 ━━━━━━━━━━━━━━━━━━━━ 4:37 1s/step - accuracy: 0.5469 - loss: 1.1401
Epoch 8: val_accuracy did not improve from 0.76923
```

```
Epoch 5/48
225/225 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.5107 - loss: 1.2783
Epoch 5: val_accuracy improved from 0.42869 to 0.47230, saving model to ./model.keras
225/225 ━━━━━━━━━━━━━━━━━━━━ 307s 1s/step - accuracy: 0.5107 - loss: 1.2782 - val_accuracy: 0.4723 - val_loss: 1.3299 - learning_rate: 0.0010
Epoch 6/48
  1/225 ━━━━━━━━━━━━━━━━━━━━ 4:29 1s/step - accuracy: 0.5312 - loss: 1.1546
Epoch 6: val_accuracy improved from 0.47230 to 0.76923, saving model to ./model.keras
225/225 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5312 - loss: 1.1546 - val_accuracy: 0.7692 - val_loss: 1.0354 - learning_rate: 0.0010
Epoch 7/48
225/225 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.5507 - loss: 1.1813
Epoch 7: val_accuracy did not improve from 0.76923
225/225 ━━━━━━━━━━━━━━━━━━━━ 574s 3s/step - accuracy: 0.5507 - loss: 1.1813 - val_accuracy: 0.5091 - val_loss: 1.2525 - learning_rate: 0.0010
Epoch 8/48
  1/225 ━━━━━━━━━━━━━━━━━━━━ 4:37 1s/step - accuracy: 0.5469 - loss: 1.1401
Epoch 8: val_accuracy did not improve from 0.76923
225/225 ━━━━━━━━━━━━━━━━━━━━ 1s 531us/step - accuracy: 0.5469 - loss: 1.1401 - val_accuracy: 0.7308 - val_loss: 0.9967 - learning_rate: 0.0010
...
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
225/225 ━━━━━━━━━━━━━━━━━━━━ 351s 2s/step - accuracy: 0.5911 - loss: 1.0708 - val_accuracy: 0.5879 - val_loss: 1.0976 - learning_rate: 0.0010
Epoch 11: early stopping
Restoring model weights from the end of the best epoch: 8.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

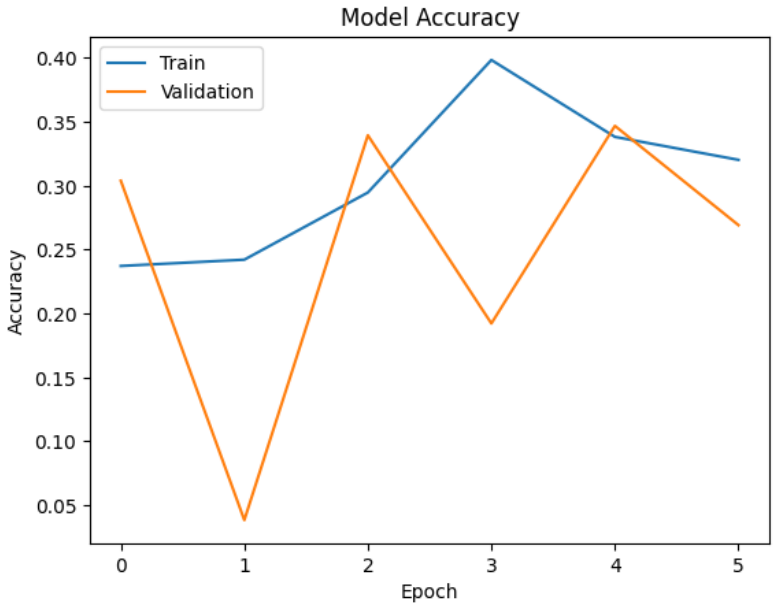
6.2 Validation metrics

- **Model Accuracy**

```
import matplotlib.pyplot as plt

# Plotting the accuracy
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

Output:



After training the model, the overall accuracy (on training and validation sets) is typically measured as the percentage of correctly classified images. This can be visualized using the history object that stores training metrics.

- **Confusion Matrix**

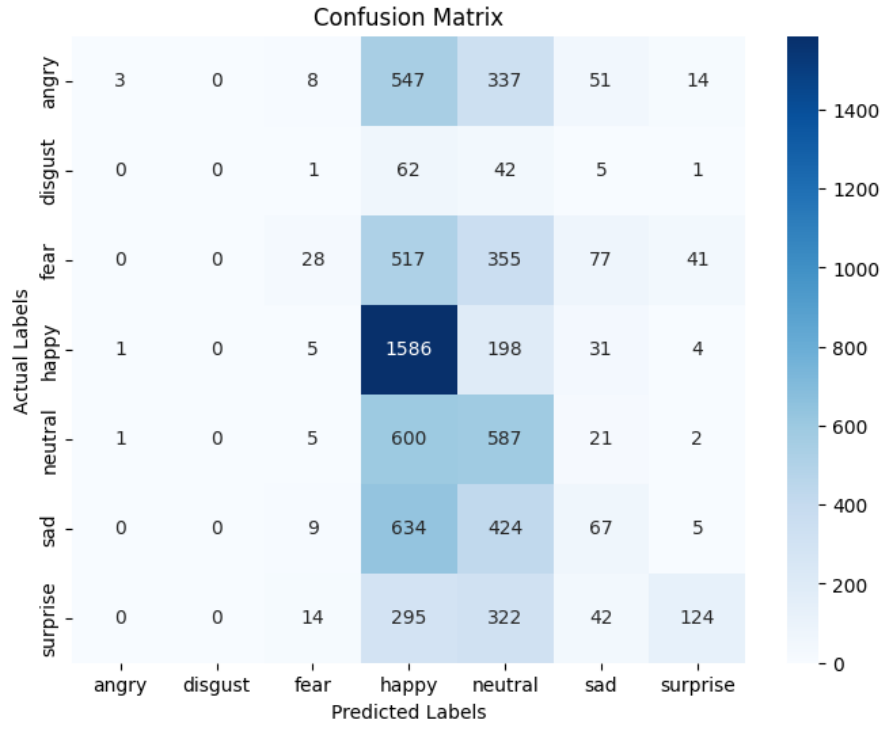
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Predicting the values
predictions = model.predict(test_set)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_set.classes
class_labels = list(test_set.class_indices.keys())

# Generating the confusion matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.ylabel('Actual Labels')
plt.xlabel('Predicted Labels')
plt.show()
```

Output



A confusion matrix helps visualize the model's performance across different emotion categories by comparing actual and predicted labels.

- Precision, Recall, and F1 Score

```
from sklearn.metrics import classification_report

# Generating the classification report
report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)
```

```
              precision    recall  f1-score   support

       angry       0.60      0.00      0.01       960
     disgust       0.00      0.00      0.00       111
        fear       0.40      0.03      0.05      1018
       happy       0.37      0.87      0.52      1825
     neutral       0.26      0.48      0.34      1216
         sad       0.23      0.06      0.09      1139
    surprise       0.65      0.16      0.25       797

    accuracy                           0.34      7066
   macro avg       0.36      0.23      0.18      7066
weighted avg       0.39      0.34      0.24      7066
```

This report will show precision, recall, and F1 score for each class (emotion) individually, as well as overall averages.

Conclusion

The emotion detection model built using deep learning techniques effectively identifies and classifies emotions from facial expressions. By employing a convolutional neural network (CNN), the model learns to extract essential features from facial images that correspond to various emotions. This process involves detecting faces, extracting key regions of interest, and using those regions to predict the emotional state of a person.

Throughout the training process, the model demonstrated to recognize emotions by capturing subtle variations in facial features from a dataset. The use of several layers in the CNN, including convolutional, pooling, and dense layers, allows the model to learn both high-level and fine-grained patterns that differentiate between emotions. Moreover, techniques such as data augmentation were employed to improve the model's robustness, helping it generalize better to unseen data.

By utilizing grayscale images and focusing on essential regions of the face, the model simplifies the input data. Furthermore, the use of emotion labels like anger, happiness, sadness, and surprise, provides clear and actionable insights into the emotions present in the facial images.

This system has significant potential in real-world applications, including enhancing human-computer interaction, improving mental health monitoring, and even assisting in areas like education, marketing, and security. It offers an automated approach to understanding emotions, which could be applied to a wide range of industries where emotional awareness plays a crucial role in decision-making and user experience.

In conclusion, the emotion detection model offers a reliable approach to recognizing and interpreting emotions from facial expressions, paving the way for future developments in affective computing and emotional intelligence technologies. With further refinements, the model could become an integral part of systems designed to better understand and respond to human emotions in various contexts.

Git-hub repository: https://github.com/Aditi200212/EmotiQuotient

Appendix

Pseudo code:

Python file (for output):

```
from keras.models import load_model
from keras.preprocessing.image import img_to_array
import cv2
import numpy as np
import os
face_classifier = cv2.CascadeClassifier(r'C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\.ipynb_checkpoints\haarcascade_frontalface_default.xml')
classifier = load_model(r'C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\.ipynb_checkpoints\model.h5')
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']
image_path = r'C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\.ipynb_checkpoints\images\e.jpeg'
img = cv2.imread(image_path)
if img is None:
    print(f"Failed to load image: {image_path}")
else:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # Detect faces in the image
    faces = face_classifier.detectMultiScale(gray)
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 255), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)
        if np.sum([roi_gray]) != 0:
            roi = roi_gray.astype('float') / 255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi, axis=0)
            prediction = classifier.predict(roi)[0]
            label = emotion_labels[prediction.argmax()]
            label_position = (x, y)
            cv2.putText(img, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        else:
            cv2.putText(img, 'No Faces', (30, 80), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    # Display the image with the emotion label
    cv2.imshow('Emotion Detector', img)
    cv2.waitKey(0)  # Wait for a key press to close the window
cv2.destroyAllWindows
```

Training and Validation data file

```
# Importing Libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout, GlobalAveragePooling2D, Flatten, Conv2D, BatchNormalization, Activation, MaxPooling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
picture_size = 48
folder_path = r"C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\train"
```

```
# Training and Validation Data
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
picture_size = 48
batch_size = 128
folder_path = r"C:\Users\BUSIREDDY ADITI\Desktop\EmotionDetection\train"
train_dir = os.path.join(r"C:\Users\BUSIREDDYADITI\Desktop\EmotionDetection\train\images\images\train")
validation_dir =os.path.join(r"C:\Users\BUSIREDDYADITI\Desktop\EmotionDetection\train\images\images\validation")
print(f"Training directory: {train_dir}")
print(f"Validation directory: {validation_dir}")
if not os.path.exists(train_dir):
    raise FileNotFoundError(f"Training directory does not exist: {train_dir}")
if not os.path.exists(validation_dir):
   raise FileNotFoundError(f"Validation directory does not exist: {validation_dir}")
datagen_train = ImageDataGenerator()
datagen_val = ImageDataGenerator()
train_set = datagen_train.flow_from_directory(
    train_dir,
    target_size=(picture_size, picture_size),
    color_mode="grayscale",
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True
)
test_set = datagen_val.flow_from_directory(
    validation_dir,
    target_size=(picture_size, picture_size),
    color_mode="grayscale",
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

```
# Model Building
from keras.models import Sequential
from keras.layers import Conv2D, BatchNormalization, Activation, MaxPooling2D, Dropout, Flatten, Dense
from keras.optimizers import Adam
no_of_classes = 7
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (5, 5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25)

model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(no_of_classes, activation='softmax'))
opt = Adam(learning_rate=0.0001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
# Fitting the Model with Training and Validation Data
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
checkpoint = ModelCheckpoint("./model.keras", monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
early_stopping = EarlyStopping(
    monitor='val_loss',
    min_delta=0,
    patience=3,
    verbose=1,
    restore_best_weights=True
)
reduce_learningrate = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=3,
    verbose=1,
    min_delta=0.0001
)
epochs = 48
history = model.fit(
    train_set,
    steps_per_epoch=train_set.n // train_set.batch_size,
    epochs=epochs,
    validation_data=test_set,
    validation_steps=test_set.n // test_set.batch_size,
    callbacks=[checkpoint, early_stopping, reduce_learningrate]
)
```