2025

DATE:
24th FEBRUARY

PRESENTED TO:
Ms. FOUQIA ZAFEER

COMPUTER VISION

PREPARED BY:

HAMMAD BIN TARIQ (BIT21203)
HUZAIFA AMIN (BIT21232)
M. ZUBAIR (BIT21250)
M. HAMZA RASHID (BIT21255)

CERTIFICATE

This is to certify that the project titled
"Facial Expression Detection"
has been completed by the following students:

Group Leader: Hammad Bin Tariq

Project Members:

Name:	Roll no:	Email:
Hammad bin Tariq	BIT-21203	**hammadtariq0007@gmail.com**
Huzaifa Amin	BIT-21232	**cocomillon7925@gmail.com**
Muhammad Zubair	BIT-21250	**mzubairbit21250@gmail.com**
M.Hamza Rashid	BIT-21255	**hamzabinrashid55@gmail.com**

of the Seventh Semester, Bachelor of information technology in the year 2025 in partial fulfillment of the requirement to the award of course **"COMPUTER VISION"**

Project submitted to:
Ms. Fouqia Zafeer
University of the Punjab, Gujranwala campus
Date: February 24,2025

Table of Contents

Documentation for Facial Expression Recognition Project

---

1. Project Overview

1.1 Project Title

Facial Expression Recognition System

1.2 Objective

The objective of this project is to develop a system that can detect human facial expressions from images or real-time video feeds. The system first detects faces using the Haar Cascade classifier and then classifies facial expressions using a pre-trained deep learning model.

1.3 Scope

- The system will identify seven distinct facial expressions: Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise.
- The system will support both static images and real-time video feeds.
- The application will be implemented using Python with TensorFlow, Keras, and OpenCV.

---

2. System Architecture

2.1 Components

Face Detection:

- **Technology**: OpenCV Haar Cascade Classifier
- **Description**: The system uses Haar Cascades to detect faces in the input image or video frame.

Expression Recognition:

- **Technology**: TensorFlow, Keras Model, OpenCV
- **Description**: Detected faces are passed to a deep learning model trained to classify the face into one of seven expressions.

2.2 Workflow

1. **Input**: User provides an image or a real-time video feed.
2. **Face Detection**: Haar Cascade detects faces in the input.
3. **Preprocessing**: Detected faces are resized and normalized.

4. **Prediction**: The preprocessed faces are fed to the model to predict the facial expression.
5. **Output**: The expression label is displayed on the image or video frame.

---

3. Setup Instructions

3.1 Environment Setup

Install Python and Required Libraries:

bash
Copy code
pip install tensorflow opencv-python numpy keras

3.2 Project Structure

bash
Copy code project-
root/

```
├── model/
│   └── model.h5   # Trained Keras model
│
├── haarcascades/
│   └── haarcascade_frontalface_default.xml                    # Haar Cascade for face detection
│
├── static_test.py   # Python script for testing on static images
└── video_test.py              # Python script for real-time video feed
```

---

4. Model Training

4.1 Dataset

The model was trained on a dataset containing labeled images of facial expressions. The dataset was preprocessed to resize images to 48x48 pixels and convert them to grayscale.

4.2 Model Architecture

python Copy
code
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout model = Sequential()

# Convolutional layers
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))

```
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.25))

# Fully connected layers model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

4.3 Training Process

```
python Copy
code
# Assuming `train_generator` and `validation_generator` are created from the dataset
model.fit(train_generator, validation_data=validation_generator, epochs=100,
steps_per_epoch=224, validation_steps=56) model.save('model.h5')
```

---

5. Code Explanation

5.1 Static Image Testing (static_test.py)

This script detects facial expressions from a static image.

```
python Copy
code import
cv2
import numpy as np
from keras.models import load_model

# Load the pre-trained model model =
load_model('model.h5')

# Load Haar Cascade for face detection
faceDetect = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Dictionary mapping numeric labels to expressions
label_dict = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Neutral', 5:'Sad', 6:'Surprise',}

# Read the input image
input = cv2.imread('face2.jpg')

# Convert the image to grayscale
gray = cv2.cvtColor(input, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = faceDetect.detectMultiScale(gray, 1.3, 1)
```

```
# Loop through detected faces for x, y, w, h in
faces:
        sub_face = gray[y:y+h, x:x+w]
        resized = cv2.resize(sub_face, (48, 48)) normalize = resized / 255.0
        reshaped = np.reshape(normalize, (1, 48, 48, 1))

        # Predict the expression
        result = model.predict(reshaped) label =
        np.argmax(result, axis=1)[0] print(label_dict[label])

        # Draw a rectangle around the face and label it with the expression cv2.rectangle(input, (x, y), (x+w, y+h),
        (0, 0, 255), 1) cv2.putText(input,          label_dict[label],          (x,          y-10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

# Display the result cv2.imshow("Frame",
input) cv2.waitKey(0)
cv2.destroyAllWindows()
```

Key Points:

- **Face Detection**: The Haar Cascade classifier detects faces in the image.
- **Preprocessing**: Detected faces are resized to 48x48 pixels and normalized by dividing pixel values by 255.
- **Prediction**: The preprocessed face is passed to the model, which predicts the expression.
- **Output**: The expression is displayed on the image with a bounding box around the face.

5.2 Real-time Video Testing (video_test.py)

This script detects facial expressions from a real-time video feed using a webcam.

```
python Copy
code import
cv2
import numpy as np
from keras.models import load_model

# Load the pre-trained model model =
load_model('model.h5')

# Initialize video capture from webcam video =
cv2.VideoCapture(0)

# Load Haar Cascade for face detection
faceDetect = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Dictionary mapping numeric labels to expressions
label_dict = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Neutral', 5:'Sad', 6:'Surprise',}

# Start the video capture loop while True:
        ret, input = video.read()
                gray = cv2.cvtColor(input, cv2.COLOR_BGR2GRAY)
```

```
    faces = faceDetect.detectMultiScale(gray, 1.3, 3)

    # Loop through detected faces for x, y, w, h in
    faces:
        sub_face = gray[y:y+h, x:x+w]
        resized = cv2.resize(sub_face, (48, 48)) normalize = resized /
        255.0
        reshaped = np.reshape(normalize, (1, 48, 48, 1))

        # Predict the expression
        result = model.predict(reshaped) label =
        np.argmax(result, axis=1)[0] print(label_dict[label])

        # Draw a rectangle around the face and label it with the expression cv2.rectangle(input, (x, y), (x+w,
        y+h), (0, 0, 255), 1) cv2.putText(input,                label_dict[label],    (x,
                                                y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

    # Display the video frame cv2.imshow("Frame",
    input)

    # Break the loop on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'): break

# Release the video capture and close windows video.release()
cv2.destroyAllWindows()
```

Key Points:

- **Real-time Processing**: The script captures video frames from the webcam and processes each frame in real-time.
- **Face Detection**: Faces are detected in each frame using the Haar Cascade classifier.
- **Expression Recognition**: The detected faces are passed to the model for expression prediction.
- **Output**: The expression is displayed on the video frame with a bounding box around the face.

---

6. Usage Instructions

6.1 Running Static Image Detection

To test the model on a static image, use the static_test.pyscript:

bash
Copy code
python static_test.py

This will read the specified image file, detect any faces, predict their expressions, and display the results.

6.2 Running Real-time Video Detection

To test the model in real-time using a webcam, use the video_test.pyscript:

bash
Copy code
python video_test.py

This will open a video window where you can see the live feed from your webcam with detected expressions displayed.

---

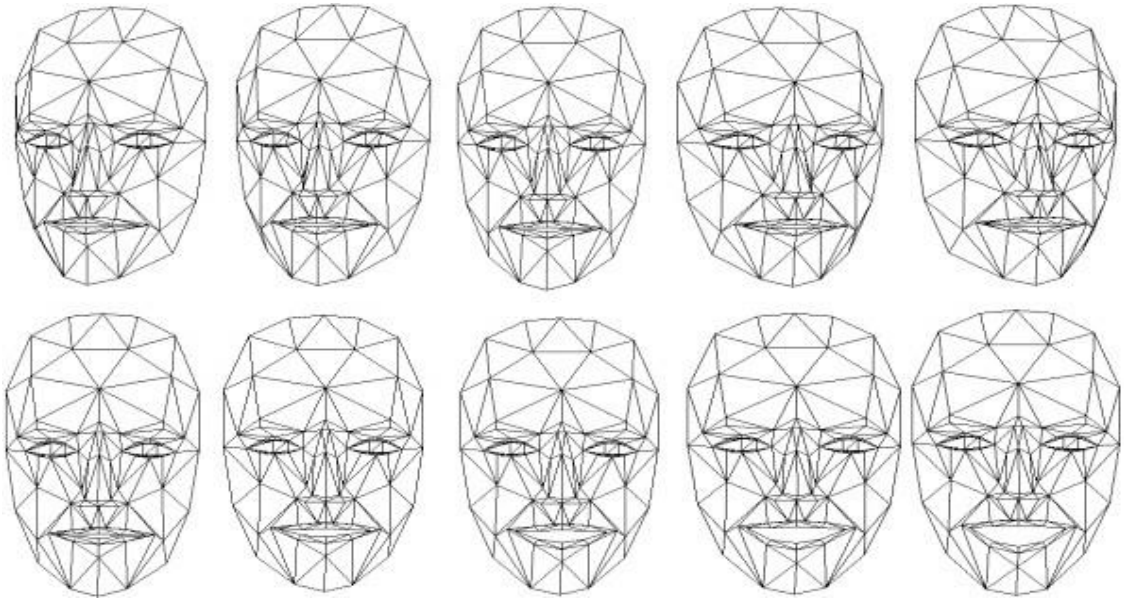
7. Model Logical Structure

7.1 Landmark Grids:

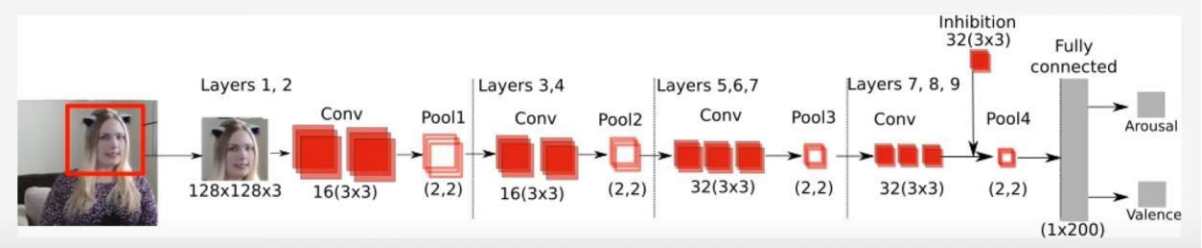
This image shows how the facial expression recognition model detects facial landmarks (key points on the face) and overlays a mesh grid on these points. The grid helps in analyzing the movement and position of facial features to determine expressions like "Laughing" and "Surprise".

7.2 Mesh Grids:

This image displays multiple faces with overlaid mesh grids, representing the facial landmarks detected by the model. These grids are used to understand the structure and movements of the facial features, which are critical for identifying different expressions.



7.3 CNN Architecture



This diagram effectively represents the flow of data through a CNN designed for facial expression recognition, from an initial face image to final predictions.