#### Department of Information Technology



**University Of The Punjab**

**Gujranwala Campus**

**Date:** 07-02-2025

**Facial Expression Detection**

Project

Documentation

#### Submitted to

**Ms. Fouqia Zafeer**

#### Submitted by

**Muhammad Sufyan (Group Lead)**

#### BIT21021 (Morning)

#### 

**University of the Punjab**

**Gujranwala Campus**

**Department of Information Technology  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Certificate

This is to certify that the project titled

**“Facial Expression Detection”**

has been completed by the following students:

|  |
| --- |
| **Group Leader:** Muhammad Sufyan |
| **Project Members:**   |  |  |  | | --- | --- | --- | | **Name:** | **Roll no:** | **Email:** | | Ali Akbar | BIT-21006 | **[aliakbar6041892@gmail.com](mailto:aliakbar6041892@gmail.com)** | | Muhammad Sufyan | BIT-21021 | **[muhammadsufyan7925@gmail.com](mailto:muhammadsufyan7925@gmail.com)** | | Muhammad Umar | BIT-21036 | **[umerfbr2003@gmail.com](mailto:umerfbr2003@gmail.com)** | | Talha Sohail | BIT-21054 | **[talhasohail5333@gmail.com](mailto:talhasohail5333@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**

**Place :** University of the Punjab, Gujranwala campus

**Date : Februray** 07,2025

**Table of Contents**

[Documentation for Facial Expression Recognition Project 1](#_bookmark0)

1. [Project Overview 1](#_bookmark1)
   1. [Project Title 1](#_bookmark2)
   2. [Objective 1](#_bookmark3)
   3. [Scope 1](#_bookmark4)
2. [System Architecture 1](#_bookmark5)
   1. [Components 1](#_bookmark6)
   2. [Workflow 1](#_bookmark7)
3. [Setup Instructions 2](#_bookmark8)
   1. [Environment Setup 2](#_bookmark9)
   2. [Project Structure 2](#_bookmark10)
4. [Model Training 2](#_bookmark11)
   1. [Dataset 2](#_bookmark12)
   2. [Model Architecture 2](#_bookmark13)
   3. [Training Process 3](#_bookmark14)
5. [Code Explanation 3](#_bookmark15)
   1. [Static Image Testing (static\_test.py) 3](#_bookmark16)
   2. [Real-time Video Testing (video\_test.py) 4](#_bookmark17)
6. [Usage Instructions 5](#_bookmark18)
   1. [Running Static Image Detection 5](#_bookmark19)
   2. [Running Real-time Video Detection 6](#_bookmark20)
7. [Model Logical Structure 6](#_bookmark21)
   1. [Landmark Grids: 6](#_bookmark22)
   2. [Mesh Grids 6](#_bookmark23)
   3. [CNN Architecture 7](#_bookmark24)
8. [Screenshots: 8](#_bookmark25)
   1. [Static Face / Image: 8](#_bookmark26)
   2. [Static Face with Graphs 8](#_bookmark27)
   3. [For Live Face 9](#_bookmark28)
   4. [Live Face with Graphs: 9](#_bookmark29)
9. [Conclusion 10](#_bookmark30)

# Documentation for Facial Expression Recognition Project

## Project Overview

### Project Title

###### Facial Expression Recognition System

### 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.

### 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.

## System Architecture

### 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.

### Workflow

###### **Input**: User provides an image or a real-time video feed.

1. **Face Detection**: Haar Cascade detects faces in the input.
2. **Preprocessing**: Detected faces are resized and normalized.

###### **Prediction**: The preprocessed faces are fed to the model to predict the facial expression.

###### **Output**: The expression label is displayed on the image or video frame.

## Setup Instructions

### Environment Setup

##### Install Python and Required Libraries:

bash

Copy code

pip install tensorflow opencv-python numpy keras

### 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

## Model Training

### 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.

### 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'])

### 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')

## Code Explanation

* 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.

* 1. **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.

## Usage Instructions

### Running Static Image Detection

To test the model on a static image, use the static\_test.py script:

bash

Copy code

python static\_test.py

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

### Running Real-time Video Detection

###### To test the model in real-time using a webcam, use the video\_test.py script:

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.

## Model Logical Structure

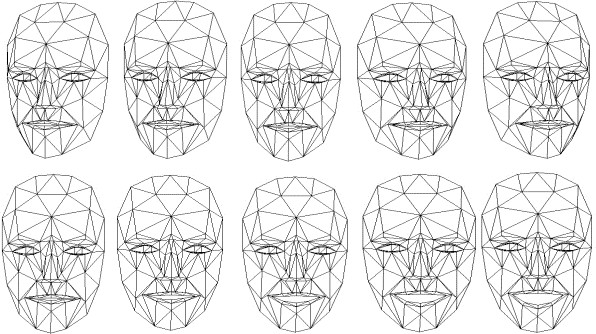
### Landmark Grids:

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".



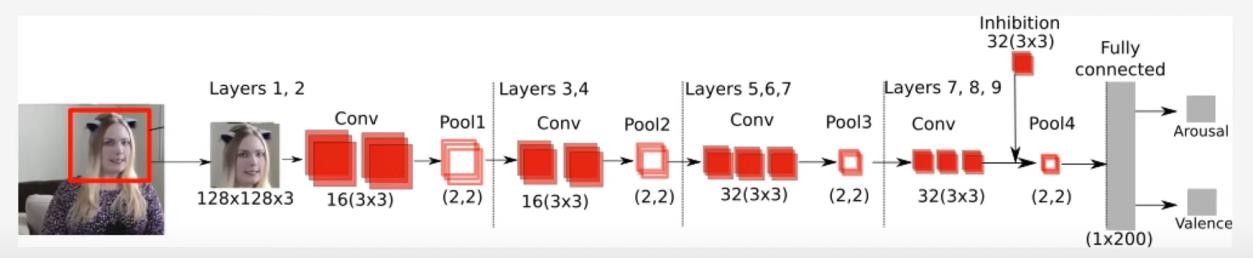
### Mesh Grids:

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.



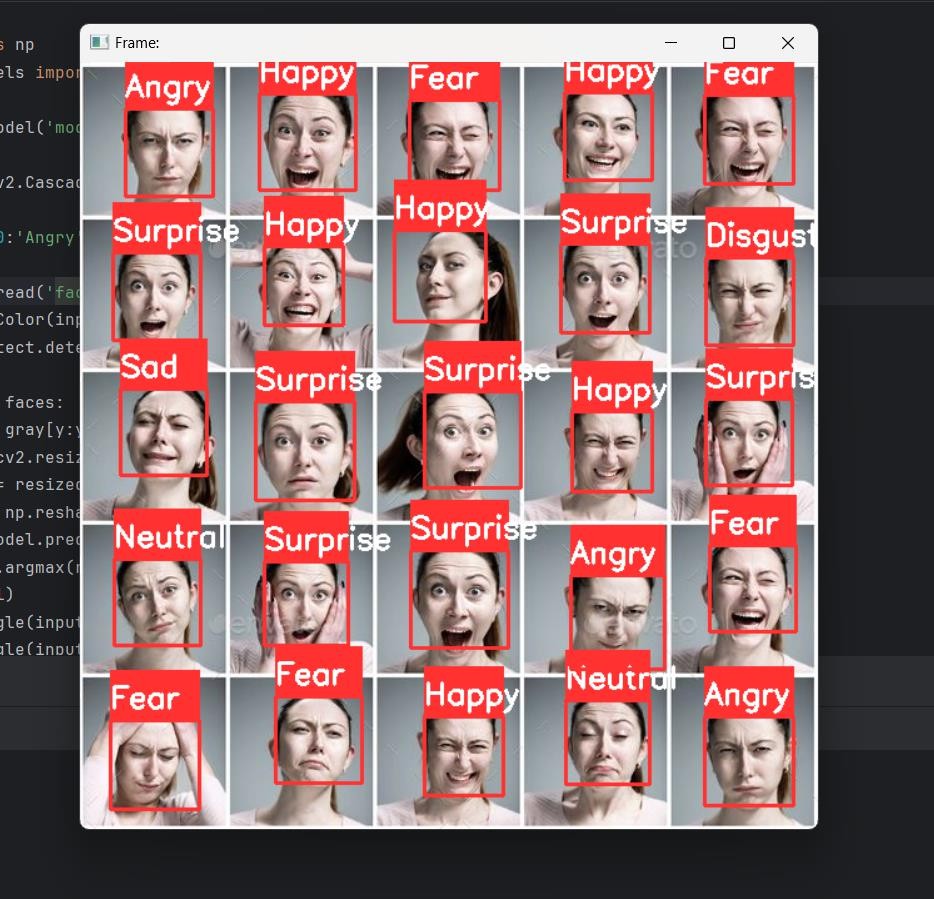
### CNN Architecture

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

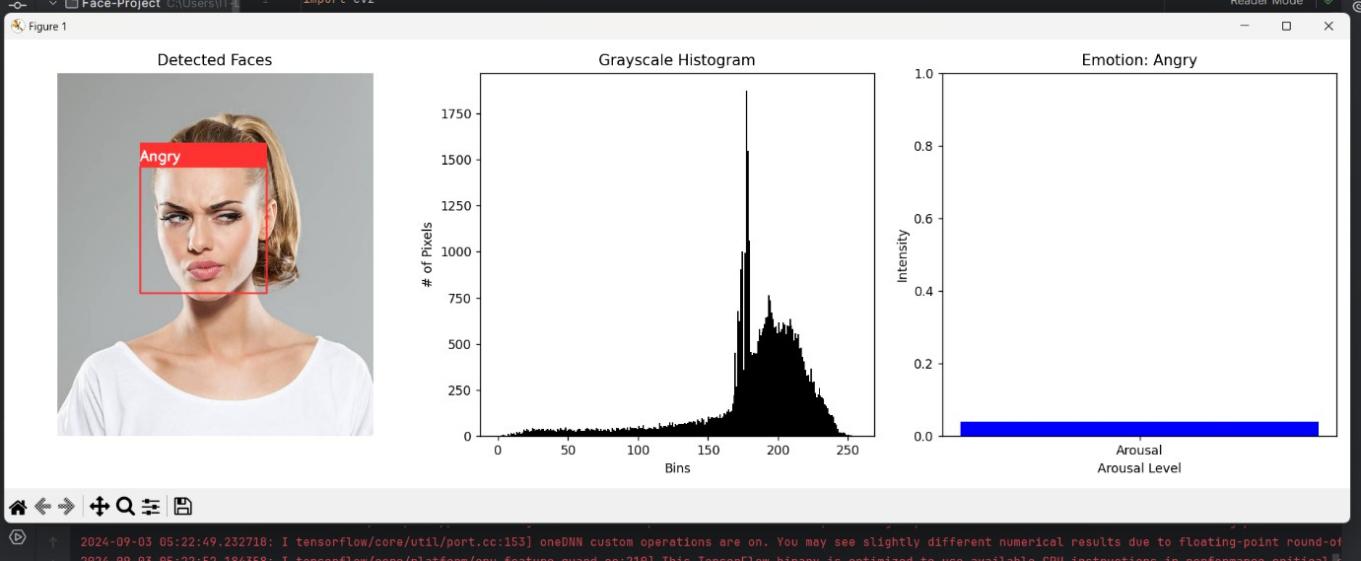


## Screenshots:

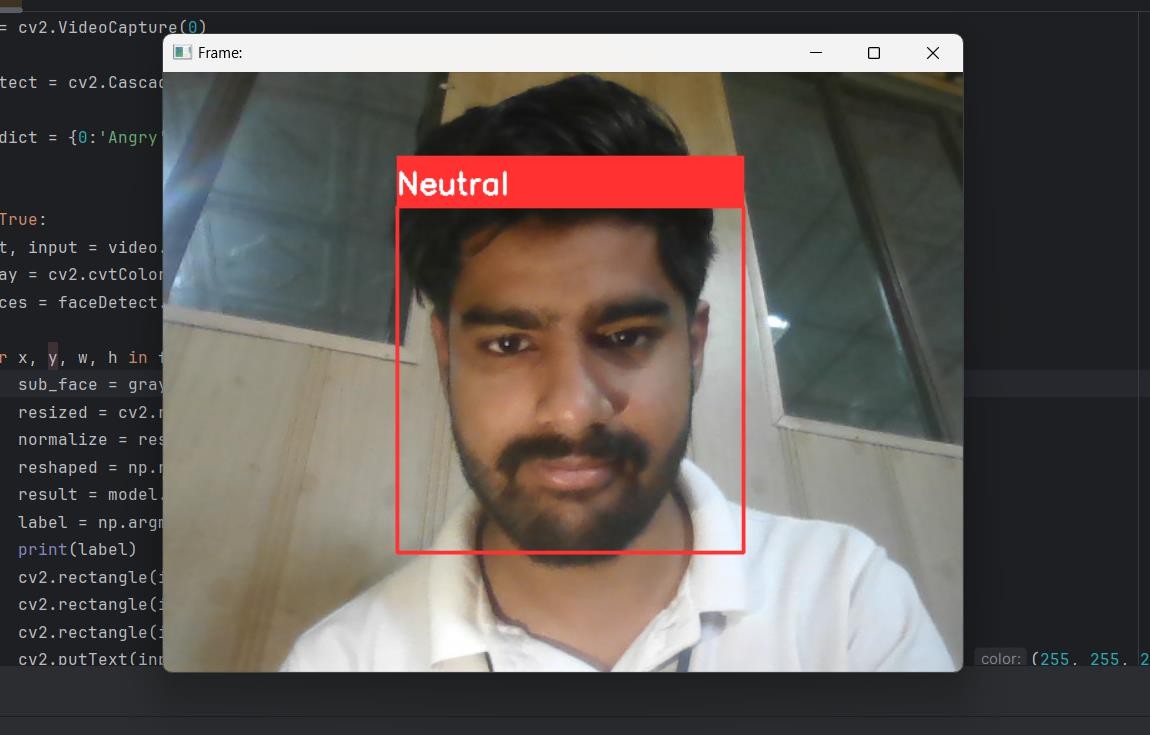
### Static Face / Image:

****

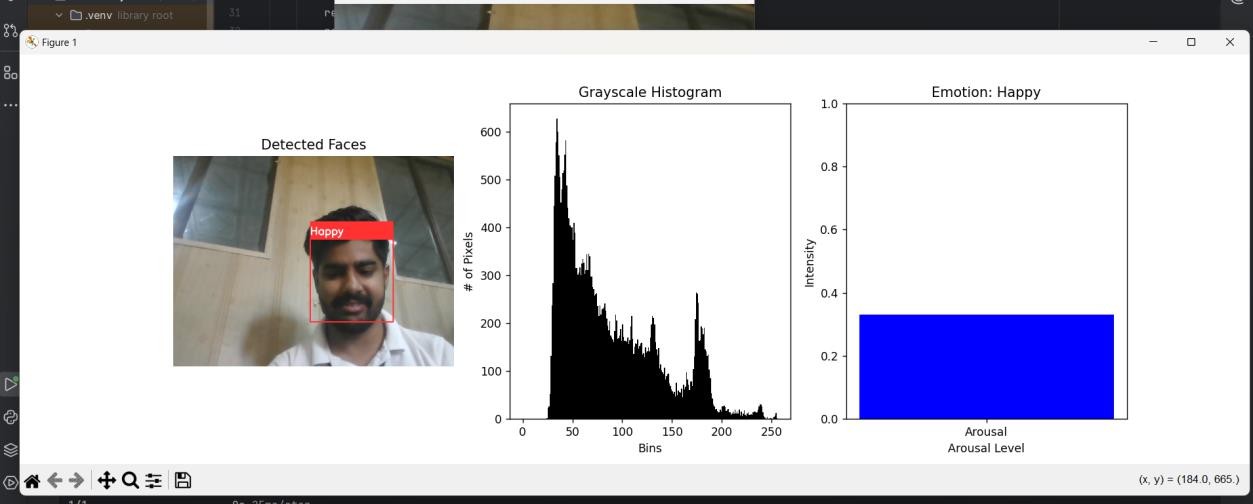
### Static Face with Graphs:

****

### For Live Face:

****

### Live Face with Graphs:

****

## Conclusion

###### This documentation provides a complete overview of the facial expression recognition project. The system is capable of detecting and classifying facial expressions from both static images and real-time video feeds. The project uses Python, TensorFlow, Keras, and OpenCV to achieve this functionality. Further improvements could include optimizing the model, expanding the range of detectable expressions, or integrating this system into a larger application.