FACIAL EMOTION RECOGNITON

AI MINI PROJECT

NAME	ROLL NO	E-MAIL
SAMEER VENU GOPAL SUNKADA	B201016EC	sameer_b201016ec@nitc.ac.in
VARKOLU VIJAY KRISHNA	B200994EC	varkolu_b200994ec@nitc.ac.in

INTRODUCTION:

Facial emotion recognition is a rapidly growing field of research that aims to automatically detect human emotions based on facial expressions. With advancements in computer vision and deep learning, it has become possible to develop highly accurate and efficient systems for facial emotion recognition. In this project, we have implemented a facial emotion recognition system using Convolutional Neural Networks (CNN) and the OpenCV (CV2) library for live facial emotion detection. We have also created a web-based user interface using HTML and Flask, which allows users to input images or access the live video stream from their camera for real-time emotion recognition.

The motivation for this project arises from the increasing demand for emotion recognition technology in various domains. For instance, in marketing research, understanding consumer preferences and emotional responses is crucial for developing effective marketing strategies. In mental health diagnosis, assessing emotional well-being can aid in identifying potential mental health issues. Additionally, in human-computer interaction, integrating facial emotion recognition can enable more intuitive and personalized interfaces. By developing a facial emotion recognition system that accurately and efficiently detects emotions from facial expressions in real-time, we aim to contribute to the advancement of this field and explore potential practical applications.

METHODOLOGY:

Dataset:

The data used in our project for training and evaluating the facial emotion recognition system consists of a labeled dataset of facial images. The dataset was obtained from a publicly available source Kaggle. The dataset comprises a diverse range of facial images depicting various emotions, such as happy, sad, angry, disgusted, surprised, fearful, and neutral expressions.

Dataset URL : https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset

PRE-PROCESSING:

In our project, we performed several preprocessing steps on the facial image data before feeding it into the CNN model for training and evaluation. The following preprocessing techniques were applied using the ImageDataGenerator class from the Keras library:

1) **Rescaling**: The pixel values of the facial images were rescaled by dividing them by 255. This was done using the rescale parameter set to 1./255 in the ImageDataGenerator. This step standardizes the pixel values to a range between 0 and 1, making it easier for the model to learn from the data.

2) **Shear Range**: Augmentation technique such as shear range was applied using the shear_range parameter set to 0.2. This introduces

shearing transformations to the facial images, adding diversity and variability to the dataset.

3) **Zoom Range**: Augmentation technique such as zoom range was applied using the zoom_range parameter set to 0.2. This introduces zooming transformations to the facial images, further augmenting the dataset with different scales and perspectives.

4) **Horizontal Flip**: Augmentation technique such as horizontal flip was applied using the horizontal_flip parameter set to True. This introduces horizontal flipping of facial images, creating mirrored images and adding more diversity to the dataset.

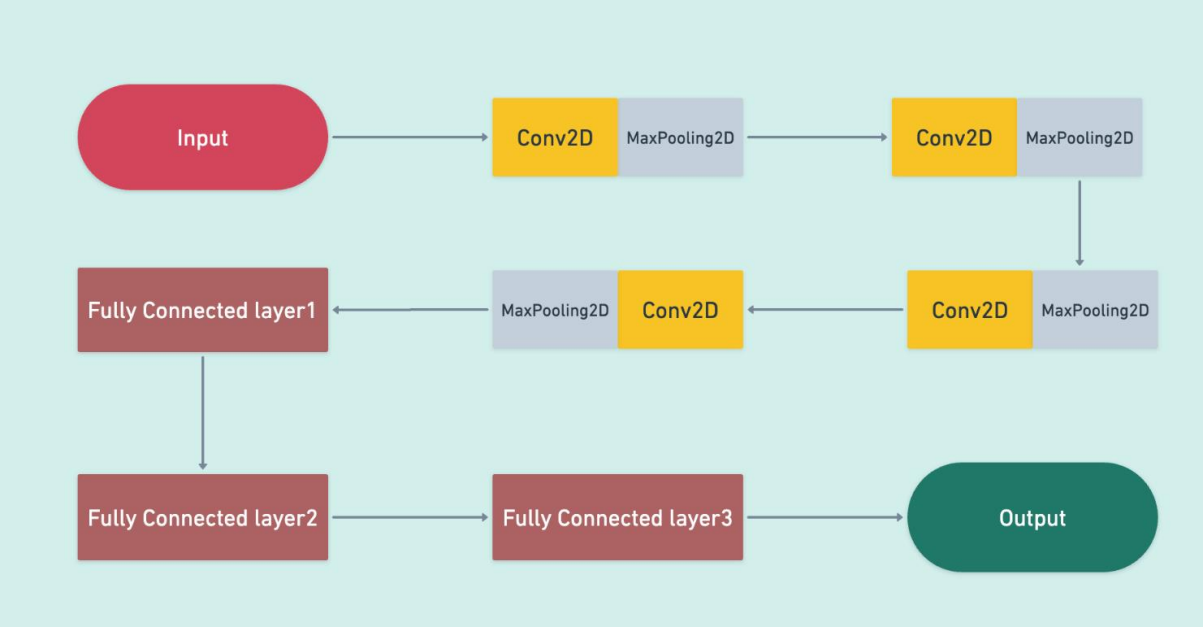
5) **Target Size and Color Mode**: The facial images in both the training and validation sets were resized to a consistent target size of (pic_size, pic_size) using the target_size parameter. Additionally, the color mode was set to "grayscale" using the color_mode parameter to convert the images to grayscale before feeding them into the model.

6) **Batch Size**: The batch size for both the training and validation sets was set to 128 using the batch_size parameter. This determines the number of images processed in each iteration during model training, balancing the trade-off between computational efficiency and model convergence.

7) **Class Mode**: The class_mode parameter was set to 'categorical' for both the training and validation sets, as we used one-hot encoding for the emotion labels associated with each facial image.

8) **Shuffle**: The shuffle parameter was set to True for the training set, which shuffles the images after each epoch during model training, helping to introduce randomness and avoid overfitting. For the validation set, the shuffle parameter was set to False to keep the images in the same order during model evaluation.

CNN MODEL ARCHITECTURE:



1) **Conv2D: Convolutional Layer:**

- Conv2D stands for 2-dimensional convolutional layer, which performs convolution operation on the input data.
- It is the fundamental building block of CNN and is responsible for extracting features from the input images.
- Parameters used: number of filters (representing the number of features to be learned), filter size (representing the size of the convolutional kernel), padding (to add zero padding to the input), and input shape (representing the shape of the input data).

2) Batch Normalization:

- Batch normalization is a technique used to improve the training stability and accelerate convergence of deep neural networks.
- It normalizes the input data by scaling and shifting it, which helps in reducing the internal covariate shift and improves the overall performance of the model.

3) Activation:

- Activation functions introduce non-linearity into the neural network, allowing it to learn complex patterns and make non-linear predictions.
- Common activation functions used in CNNs include ReLU (Rectified Linear Unit), which is used in this code, as well as sigmoid, tanh, and SoftMax.
- MaxPooling2D: Max Pooling Layer
- Max pooling is a down sampling operation that reduces the spatial dimensions of the input data.
- It helps in reducing the computational complexity of the model and retains the most important features from the input data by selecting the maximum value from a group of values (usually in a 2x2 or 3x3 window).

4) Dropout:

- Dropout is a regularization technique used to prevent overfitting in neural networks.
- It randomly sets a fraction of the input units to 0 at each training iteration, which helps in preventing the model from relying too heavily on any input unit during training, and thus improves the generalization ability of the model.

5) Flatten:

- Flatten is a layer that converts the multi-dimensional output from the convolutional layers into a 1-dimensional array.
- It is usually used as a transition between the convolutional and fully connected layers.
- Dense: Fully Connected Layer
- Dense layer, also known as fully connected layer, connects every neuron from the previous layer to every neuron in the current layer.
- It is responsible for the final decision-making and prediction based on the learned features from the previous layers.

6) SoftMax:

- Softmax is an activation function used in the output layer for multi-class classification problems.
- It converts the final output of the model into a probability distribution over the classes, where the class with the highest probability is predicted as the final output.

7) Adam: Optimizer

- Adam (short for Adaptive Moment Estimation) is an optimization algorithm used for updating the model parameters during training.
- It is a popular optimizer that combines the advantages of both RMSprop and Momentum optimization algorithms and is commonly used in deep learning models for its fast convergence and good performance.

8) **Loss function:**

- Loss function, also known as objective or cost function, measures the error between the predicted output and the actual target labels during training.
- Categorical cross-entropy is a commonly used loss function for multi-class classification problems, as it measures the dissimilarity between the predicted class probabilities and the true class probabilities.

9) **Metrics:**

- Metrics are used to evaluate the performance of the model during training and testing.
- Accuracy is a common metric used in classification problems, which measures the proportion of correctly predicted samples out of the total samples.

Model Optimization techniques:

1) **Early Stopping:**

- Early stopping is a technique used to prevent overfitting and improve model generalization by monitoring the model's performance during training and stopping the training process early if the performance on the validation data deteriorates.
- In the code, Early Stopping is configured with the following parameters:
    - **monitor:** The quantity to be monitored during training, which is 'val_loss' in this case, i.e., the validation loss.
    - **min_delta:** The minimum change in the monitored quantity to be considered as an improvement. If the change is less than this value, it is considered as no improvement.

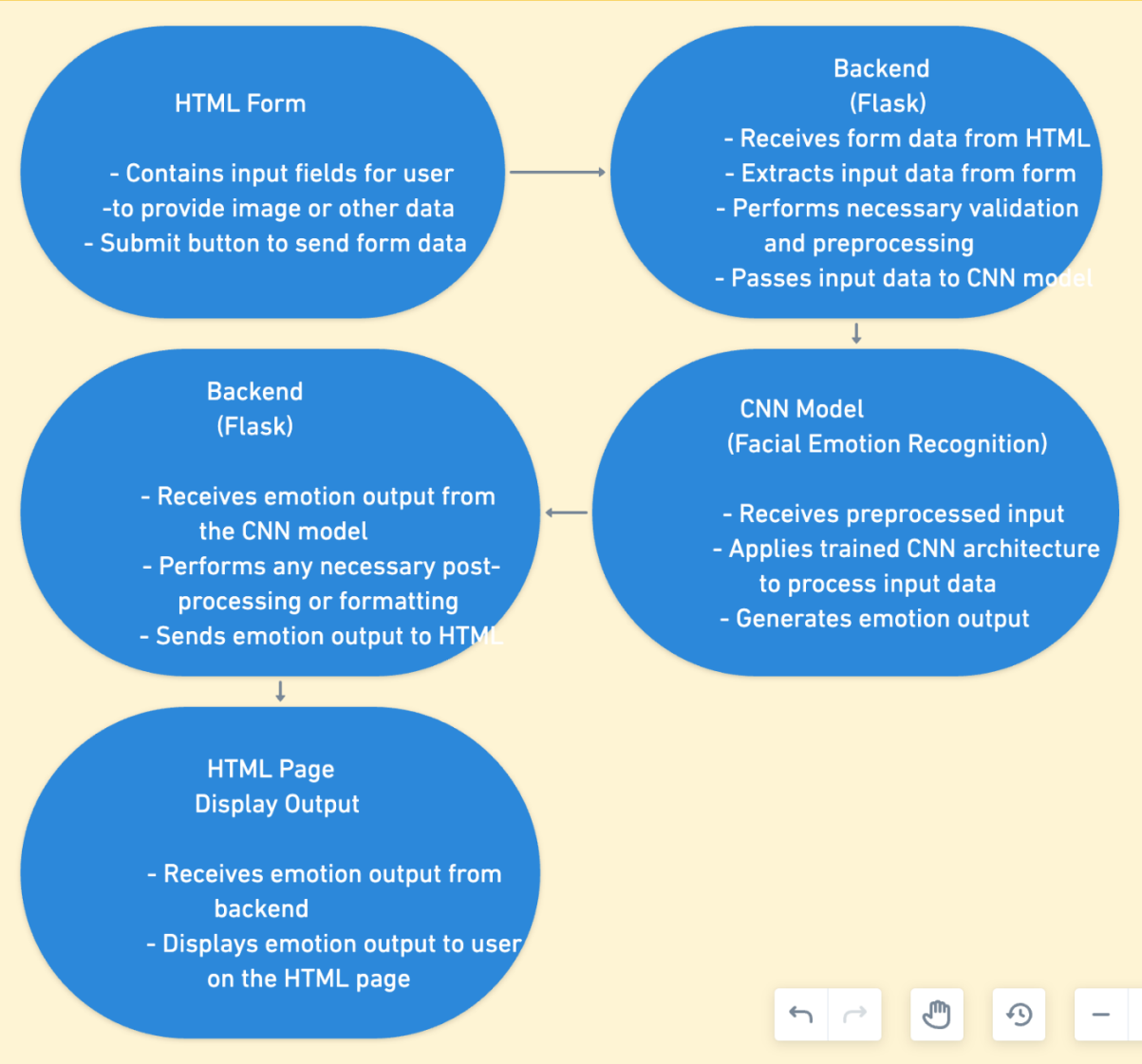
- **patience**: The number of epochs to wait for improvement in the monitored quantity before stopping the training process.
- **restore_best_weights:** If set to True, the best weights of the model based on the monitored quantity will be restored after training.

2) ReduceLROnPlateau:

- ReduceLROnPlateau is a technique used to dynamically adjust the learning rate of the optimizer during training to help the model converge faster and achieve better performance.
- In the code, ReduceLROnPlateau is configured with the following parameters:
  - **monitor**: The quantity to be monitored during training, which is 'val_loss' in this case, i.e., the validation loss.
  - **factor**: The factor by which the learning rate will be reduced when the monitored quantity stops improving.
  - **patience**: The number of epochs to wait for improvement in the monitored quantity before reducing the learning rate.
  - **min_delta**: The minimum change in the monitored quantity to be considered as an improvement. If the change is less than this value, it is considered as no improvement.

These model optimization techniques help in improving the training process of the CNN model by preventing overfitting, speeding up convergence, and achieving better generalization performance. You can mention their usage in your report as effective techniques to optimize deep learning models and improve their performance.

FACIAL EMOTION RECOGNITON WEBSITE FLOWCHART



OPENCV FLOWCHART:

CNN MODEL IMPLEMENTATION:

1) Importing and unzipping dataset:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
!kaggle datasets download -d jonathanoheix/face-expression-recognition-dataset
```

```
import zipfile
zip_ref = zipfile.ZipFile('/content/face-expression-recognition-dataset.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

2) Importing necessary libraries:

```
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import load_img, ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.models import Model,Sequential
from tensorflow.keras.layers import Conv2D, Flatten, BatchNormalization, Dense, MaxPooling2D, Activation,Dropout
```

3) Displaying a few images from dataset:

```
pic_size = 48
folder_path = "/content/images"

expression = 'happy'
plt.figure()
for i in range(1, 10, 1):
    plt.subplot(3,3,i)
    img = load_img(folder_path + '/train/' + expression + '/' +
                   os.listdir(folder_path + '/train/' +expression)[i],
                   target_size = (pic_size, pic_size))
    plt.imshow(img)
plt.show()
```

4) Data PreProcessing

```
batch_size = 128
data_train = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
data_test = ImageDataGenerator(
    rescale=1./255
)

train_set = data_train.flow_from_directory(folder_path+"/train",
                                           target_size = (pic_size,pic_size),
                                           color_mode = "grayscale",
                                           batch_size=batch_size,
                                           class_mode='categorical',
                                           shuffle=True)
test_set = data_test.flow_from_directory(folder_path+"/validation",
                                           target_size = (pic_size,pic_size),
                                           color_mode = "grayscale",
                                           batch_size=batch_size,
                                           class_mode='categorical',shuffle=False)
```

5) CNN Model:

```
no_of_classes = 7
model = Sequential()

#1st CNN layer
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))

#2nd CNN layer
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))

#3rd CNN layer
model.add(Conv2D(256,(3,3),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#4th CNN layer
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())

#Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(no_of_classes, activation='softmax'))



opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

6) Model Optimization:

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

checkpoint = ModelCheckpoint("./myfermodel_e11.h5", monitor='val_acc', verbose=1, save_best_only=True, mode='max')

early_stopping = EarlyStopping(monitor='val_loss',
                           min_delta=0,
                           patience=5,
                           verbose=1,
                           restore_best_weights=True
                           )

reduce_learningrate = ReduceLROnPlateau(monitor='val_loss',
                                factor=0.2,
                                patience=4,
                                verbose=1,
                                min_delta=0.0001)

callbacks_list = [early_stopping,checkpoint,reduce_learningrate]

epochs = 48

model.compile(loss='categorical_crossentropy',
              optimizer = Adam(lr=0.001),
              metrics=['accuracy'])
```

7) Training the model:

```
history = model.fit_generator(generator=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=callbacks_list)
```

8) Plotting the Accuracy and Loss curves:

```
plt.style.use('dark_background')

plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.show()
```

9) Evaluation Metrics:

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import numpy as np
y_pred = model.predict(test_set)
y_pred = np.argmax(y_pred, axis=1)
y_true = test_set.labels

accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1score = f1_score(y_true, y_pred, average='weighted')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1score)
```

10) Saving model as .h5 file:

```
from tensorflow.keras.models import load_model
model.save('/content/drive/MyDrive/my_models/fer_model8.h5')
```

FLASK IMPLEMENTATION:

```
from flask import Flask, request, jsonify, render_template
from PIL import Image
import numpy as np
import tensorflow as tf

app = Flask(__name__)

@app.route("/code")
def code():
    return render_template('ai_deep.html')

@app.route("/predict", methods=["POST"])
def predict():
    model = tf.keras.models.load_model("fer_model8.h5")
    image = Image.open(request.files["image"])
    image = image.convert('L')  # convert image to grayscale
    image = image.resize((48, 48))
    image = np.array(image) / 255.0
    image = np.expand_dims(image, axis=0)
    prediction = model.predict(image)
    emotion = np.argmax(prediction)
    x = ""
    emotions = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
    x = emotions[int(str(emotion))]
    return jsonify({"emotion": x})

if __name__ == "__main__":
    app.run(port='4000', debug=True)
```

HTML:

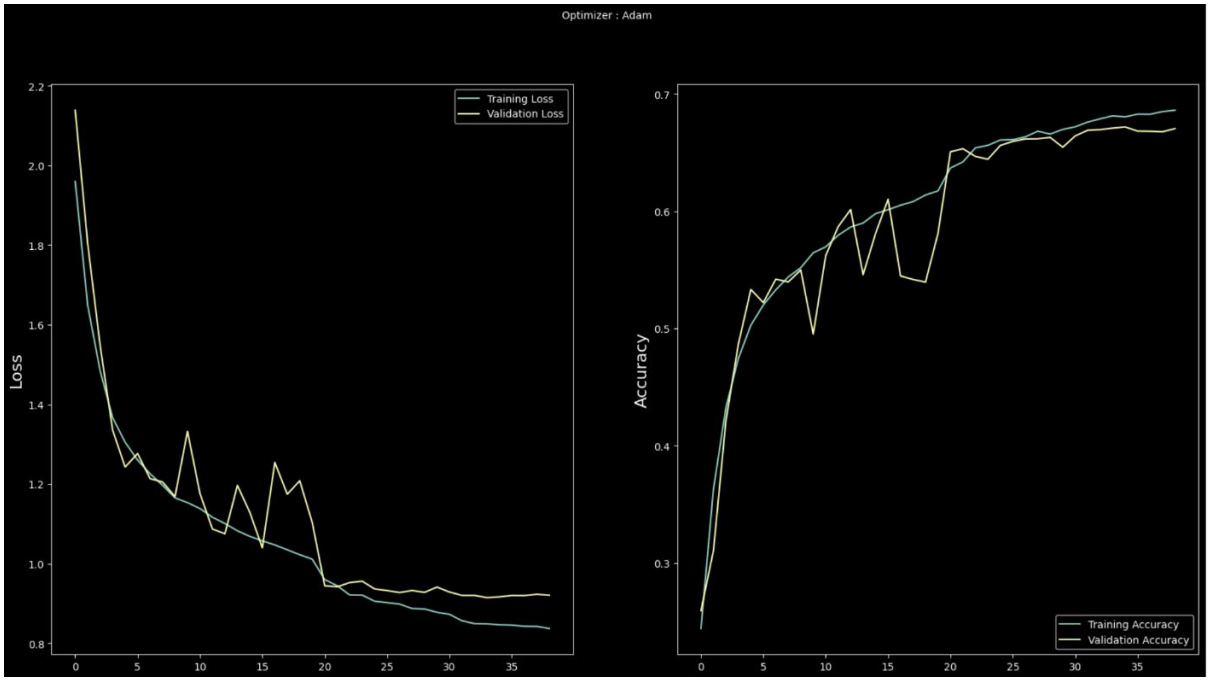
```html
<!DOCTYPE html>
<html>
  <head>
    <title>Facial Emotion Recognition</title>
    <style>
      body {
        font-family: Arial, Helvetica, sans-serif;
        margin: 0;
        padding: 0;
        background-color: #f2f2f2;
      }

      h1 {
        text-align: center;
        color: #8b0000;
        margin-top: 50px;
      }

      form {
        display: flex;
        justify-content: center;
        margin-top: 50px;
      }

      input[type="file"] {
        padding: 10px;
        border: 2px solid #8b0000;
        border-radius: 5px;
        font-size: 18px;
        margin-right: 10px;
      }

      button[type="button"] {
        padding: 10px;
        background-color: #8b0000;
        color: #fff;
        font-size: 18px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
      }

      #result {
        font-size: 40px;
        text-align: center;
        color: #8b0000;
        margin-top: 50px;
      }

      #image-display {
        max-width: 300px;
        max-height: 300px;
        display: block;
        margin: 0 auto;
        margin-top: 50px;
        border: 2px solid #8b0000;
        border-radius: 5px;
      }
    </style>
  </head>
  <body>
    <h1>Facial Emotion Recognition</h1>
    <form>
      <input type="file" id="image-input" name="image">
      <button type="button" onclick="sendImage()">Submit</button>
    </form>
    <p id="result"></p>
    <img id="image-display" src="#" alt="Image preview">
    <script>
      function sendImage() {
        const fileInput = document.getElementById("image-input");
        const file = fileInput.files[0];
        const formData = new FormData();
        formData.append("image", file);
        fetch("/predict", {
          method: "POST",
          body: formData,
        })
          .then((response) => response.json())
          .then((data) => {
            document.getElementById("result").textContent = data.emotion;
            document.getElementById("image-display").src = URL.createObjectURL(file);
          })
          .catch((error) => console.error(error));
      }
    </script>
  </body>
</html>
```

REALTIME EMOTION DETECTION USING OPENCV:

```
import os
import cv2
import numpy as np
from tensorflow.keras.preprocessing import image
import warnings
warnings.filterwarnings("ignore")
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from keras.models import  load_model
import matplotlib.pyplot as plt
import numpy as np

# load mode
model = load_model("fer_model8.h5")


face_haar_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')


cap = cv2.VideoCapture(0)

while True:
    ret, test_img = cap.read()  # captures frame and returns boolean value and captured image
    if not ret:
        continue
    gray_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY) # convert color image to grayscale

    faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.32, 5)

    for (x, y, w, h) in faces_detected:
        cv2.rectangle(test_img, (x, y), (x + w, y + h), (255, 0, 0), thickness=7)
        roi_gray = gray_img[y:y + w, x:x + h]  # cropping region of interest i.e. face area from  image
        roi_gray = cv2.resize(roi_gray, (48, 48))
        img_pixels = image.img_to_array(roi_gray)
        img_pixels = np.expand_dims(img_pixels, axis=0)
        img_pixels /= 255

        predictions = model.predict(img_pixels)

        # find max indexed array
        max_index = np.argmax(predictions[0])

        emotions = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
        predicted_emotion = emotions[max_index]

        cv2.putText(test_img, predicted_emotion, (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    resized_img = cv2.resize(test_img, (1000, 700))
    cv2.imshow('Facial emotion analysis ', resized_img)

    if cv2.waitKey(10) == ord('q'):  # wait until 'q' key is pressed
        break

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

RESULTS:

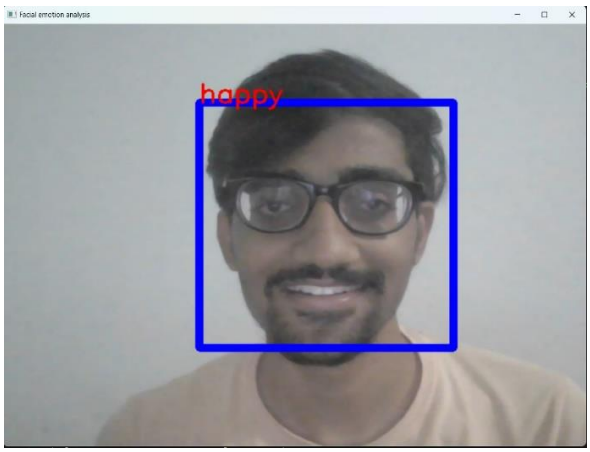
Plots of Accuracy and Loss:

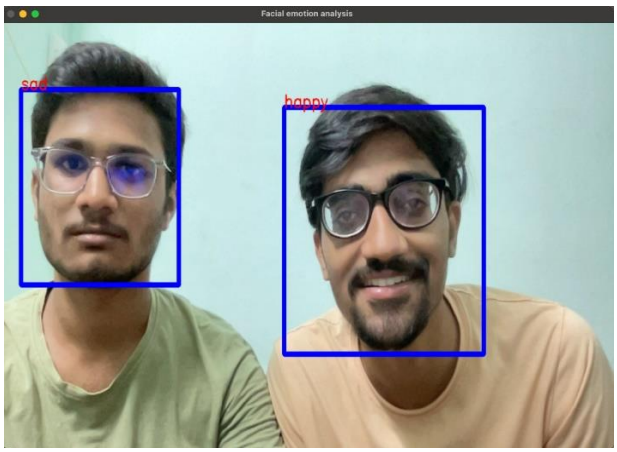


Evaluation Metrics score:

```
Accuracy: 0.6712425700537786
Precision: 0.671401441618232
Recall: 0.6712425700537786
F1 score: 0.6677883024853748
```

Real time Facial Emotion detection:

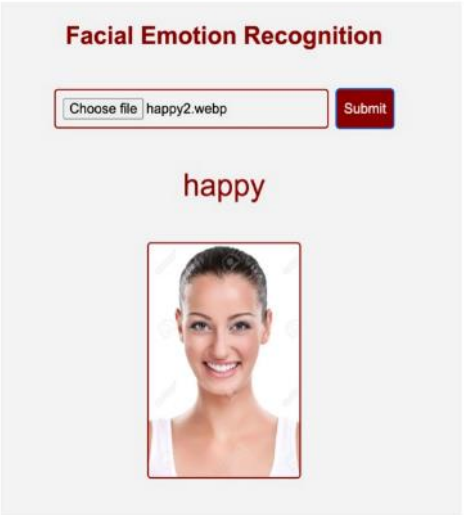




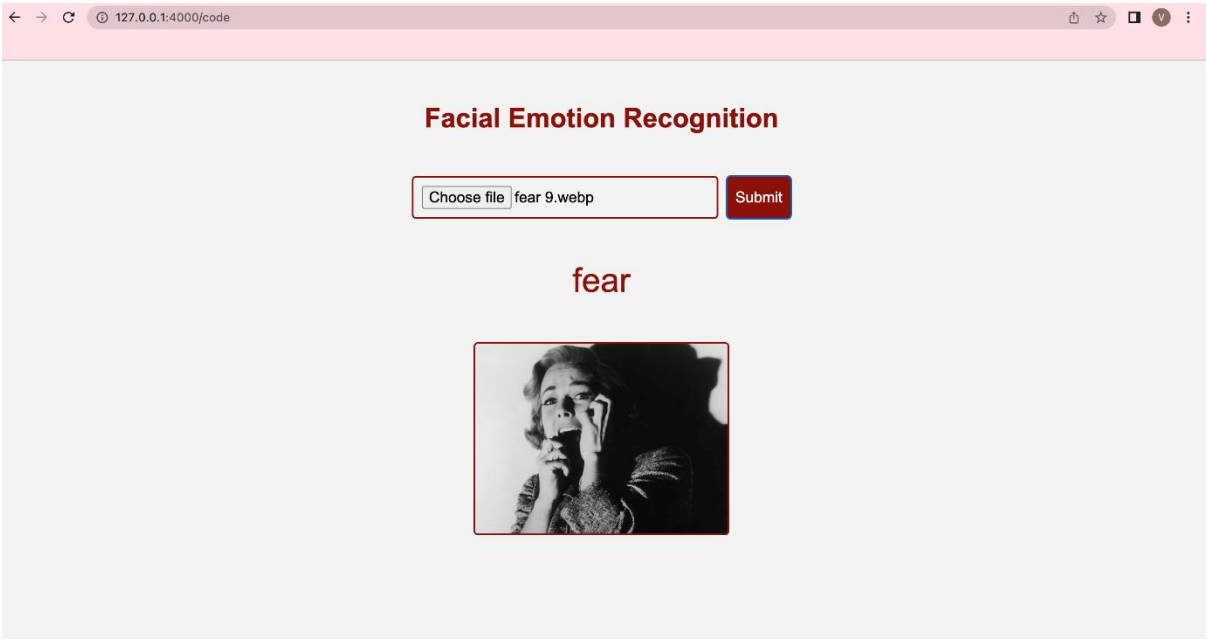
HTML output:


Facial Emotion Recognition
Choose file surprise.jpeg
Submit
surprise


Facial Emotion Recognition
Choose file sad.jpeg
Submit
sad


Facial Emotion Recognition
Choose file happy2.webp
Submit
happy


Facial Emotion Recognition
Choose file angry7.jpeg
Submit
angry


127.0.0.1:4000/code
Facial Emotion Recognition
Choose file fear 9.webp
Submit
fear

CONCLUSION:

In conclusion, the proposed facial emotion recognition (FER) system based on Convolutional Neural Networks (CNN) integrated with OpenCV for live emotion prediction from camera input, along with a web-based user interface, has shown promising results. The trained CNN model achieved a satisfactory level of accuracy on the FER dataset, and the OpenCV-based live prediction from camera input provides real-time feedback. The web-based user interface allows for convenient interaction and practical usage of the FER system. However, further optimizations and improvements can be made, such as fine-tuning the model with more data, exploring different CNN architectures, and enhancing the user interface for better usability. Overall, the FER system with OpenCV integration offers potential for various applications, including emotion-aware user interfaces, human-computer interaction, and affective computing.

REFERENCES:

1. Deep Facial Expression Recognition: A Survey Shan Li and Weihong Deng* , Member, IEEE - https://arxiv.org/pdf/1804.08348.pdf.

2. FACIAL EMOTION DETECTION USING CONVOLUTIONAL NEURAL NETWORKS by Mohammed Adnan Adil Bachelor of Engineering, Osmania University, 2016 -

https://dspace.library.uvic.ca/bitstream/handle/1828/13388/Adil_Mohammed_Adnan_MEng_2021.pdf?sequence=3