COMPUTER VISION PROJECT REPORT

MAY 2023

EMOTION DETECTION

BY:

MUJATABA ABBAS - 200901070
SYEDA FARAWA RIZVI - 200901098

TO:

SIR NADEEM

INTRODUCTION

PROBLEM STATEMENT

Develop a deep learning model that detects and classifies emotions using facial images. The goal is to, with the use of Computer Vision and deep learning algorithms, analyze facial expressions and predict emotional states with respect to this. Model is trained and evaluated on a dataset (FER-2013) which includes facial images categorized into different emotions (7 emotions).

OBJECTIVES

- To develop an accurate emotion detection system that can recognize facial expressions in real time.
- To generate an emotion report to visualize the emotional distribution.
- To use a deep learning algorithm knowns as Convolutional Neural Network (CNN) to train the model for emotion classification.
- To evaluate the performance of the emotion detection system.
- To compare the hyperparameters to identify the most effective configuration,
- To develop a user-friendly interface to showcase the emotion detection system a provide convenient usage.

SCOPE

Emotion detection allows us to understand and analyze human emotions, which are crucial points of interactions and behavior. By accurately detecting and interpreting emotions, we can gain valuable insights into psychology and social interactions. It can aid systems to respond appropriately to people.

APPLICATIONS

- This project is used in mental health fields to let the therapist and patient interactions be more enhanced and helpful in progress.
- It can be applied in any computer-user interaction to enhance its experience by understanding user emotions which can lead to a more personalized user experience.
- Market research and advertising can use to understand consumers' emotional responses to advertisements and products, then further tailor products to the customers' needs.
- Educational environments can use this to create more effective learning systems by keeping in check the students well being and emotional state during learning.

MOTIVATIONS

1. Enhanced User Experiences
2. Psychological Insight
3. Improved Decision Making
4. Improved Artificial Intelligence

LITERATURE REVIEW

Emotion recognition, an essential area in computer vision, is brimming with possibilities due to its wide array of applications.

The dawn of deep learning has sparked a transformative shift in this field. Convolutional Neural Networks (CNNs) have shown great promise, especially following the pioneering work of Goodfellow et al. and the introduction of the comprehensive FER2013 dataset.

While challenges exist in the form of varying lighting, occlusion, facial orientations, and cultural differences, these have inspired researchers to devise even more innovative solutions. One of the most exciting developments is the exploration of multimodal emotion recognition, which combines inputs like facial expressions, body language, and speech tone.

In essence, the field of emotion detection in computer vision is not only expanding rapidly but also innovating in thrilling ways, making it a promising field for further research and development.







METHODOLOGY

DATA COLLECTION - DATASET

The FER-2013 dataset was used to train the algorithm and test the algorithm.
This dataset consists of 48x48 grayscale images of faces. These faces are automatically registered, so the face is more or less centered and takes up similar space.
FER-2013 holds two folders, one for train, and one for test, each folder has further sub-folders named after every emotion.
(Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral)

The training set includes 28,709 examples.
The testing set includes 3,589 examples.
Total of 35,887 images.
It is split into 80% for training, 10% for validation, and 10% for testing.

This is highly influential in the field and has been used in many research studies to compare the performance of different facial emotion recognition algorithms and models.

With its challenges, such as dealing with the illumination, face orientation, and quality of the image.

MODEL ARCHITECTURE

- **Sequential Structure**: layers are added in sequential order from the input to the output. This sequential nature allows the model to learn and build upon the representations learned in the previous layers.
- **Hierarchical Feature Learning**: The model leverages the sequential structure to learn hierarchical representations of features. In the early layers of the model, such as the initial convolutional layers, the model learns low-level features like edges, gradients, and textures. These features capture basic visual patterns in the input images.
- **Stacking Layers**: As the model progresses, it adds more layers, such as additional convolutional and pooling layers. The model can capture more complex and abstract features by stacking these layers together. The early layers capture basic shapes and textures, while the later layers build upon these representations to learn more high-level features, such as facial expressions.
- **Spatial and Temporal Information**: The convolutional layers, with their filters and kernels, allow the model to capture spatial information from the images. They can detect patterns and features in different parts of the image. This spatial information is crucial for understanding facial expressions, as emotions are often expressed through specific facial regions.
- **Fully Connected Layers**: At the end of the model, take the flattened feature representations from the previous layers and learn to classify the input images into different emotion categories. These layers combine the learned features and perform the final decision-making based on the extracted representations.
- **Emotion Classification**: The final fully connected layer has 7 units, representing the 7 emotion categories. The softmax activation function converts the outputs of these units into probabilities, indicating the likelihood of each emotion category. The model predicts the emotion category with the highest probability as the predicted emotion for a given input image.

By combining the sequential structure, convolutional layers, pooling layers, and fully connected layers, the model learns to extract relevant spatial and temporal information from the input images. This allows the model to understand and differentiate between different facial expressions associated with various emotions, enabling it to classify the input images into the appropriate emotion categories.

The model architecture is designed to leverage the power of deep learning to automatically learn and extract features from the images, rather than relying on handcrafted features. This enables the model to adapt and generalize well to different facial expressions and emotions, making it effective for emotion detection tasks.

PERFORMANCE EVALUATION

Performance metrics will be evaluated of the trained model on the basis of accuracy and loss.
Comparison of training and validation performance.

We have developed versions of the neural network model for emotion detection, with some differences in the hyperparameters and training configuration. Let's break down the key differences and analyze the last epoch outputs while hyperparameter tuning:

Version 1:

- Hyperparameters:
  - Learning rate (lr): 0.0001
  - Batch size: 64
  - Dropout rates: 0.25 and 0.5
  - Number of epochs: 50
- Last epoch output:
  - Validation loss: 1.2261
  - Validation accuracy: 0.6223

Version 2:

- Hyperparameters:
  - Learning rate (lr): 0.0010
  - Batch size: 70
  - Dropout rates: 0.30 and 0.55
  - Number of epochs: 100
- Last epoch output:
  - Validation loss: 1.2833
  - Validation accuracy: 0.6294

Version 3:

- Hyperparameters:
  - Learning rate (lr): 0.0001
  - Batch size: 70
  - Dropout rates: 0.30 and 0.55
  - Number of epochs: 100
- Last epoch output:
  - Validation loss: 1.3186
  - Validation accuracy: 0.6371

Version 4:

- Hyperparameters:
  - Learning rate (lr): 0.0001
  - Batch size: 50
  - Dropout rates: 0.35 and 0.60
  - Number of epochs: 100
- Last epoch output:
  - Validation loss: 1.1666
  - Validation accuracy: 0.6431

Version 5: FINAL

- Hyperparameters:
  - Batch size: 32 (reduced from previous versions)
- Last epoch output:
  - Validation loss: 0.954
  - Validation accuracy: 0.6454

Version 6:

- Hyperparameters:
  - Batch size: 32
  - Additional data augmentation techniques:
    - Rotation range: Randomly rotate the images by 20 degrees
    - Width shift range: Randomly shift the width of the images by 20%
    - Height shift range: Randomly shift the height of the images by 20%
    - Shear range: Randomly apply shearing transformations
    - Zoom range: Randomly zoom in and out on the images
    - Horizontal flip: Randomly flip the images horizontally
- Last epoch output:
  - Validation loss: 0.9578
  - Validation accuracy: 0.6385

Version 7:

- Dataset:
  - Training: reduced to 19126
  - Validation: reduced to 6781
- Hyperparameters:
  - Batch size: 32
  - epochs: 100
- Last epoch output:
  - Validation loss: 1.1248
  - Validation accuracy:0.6391

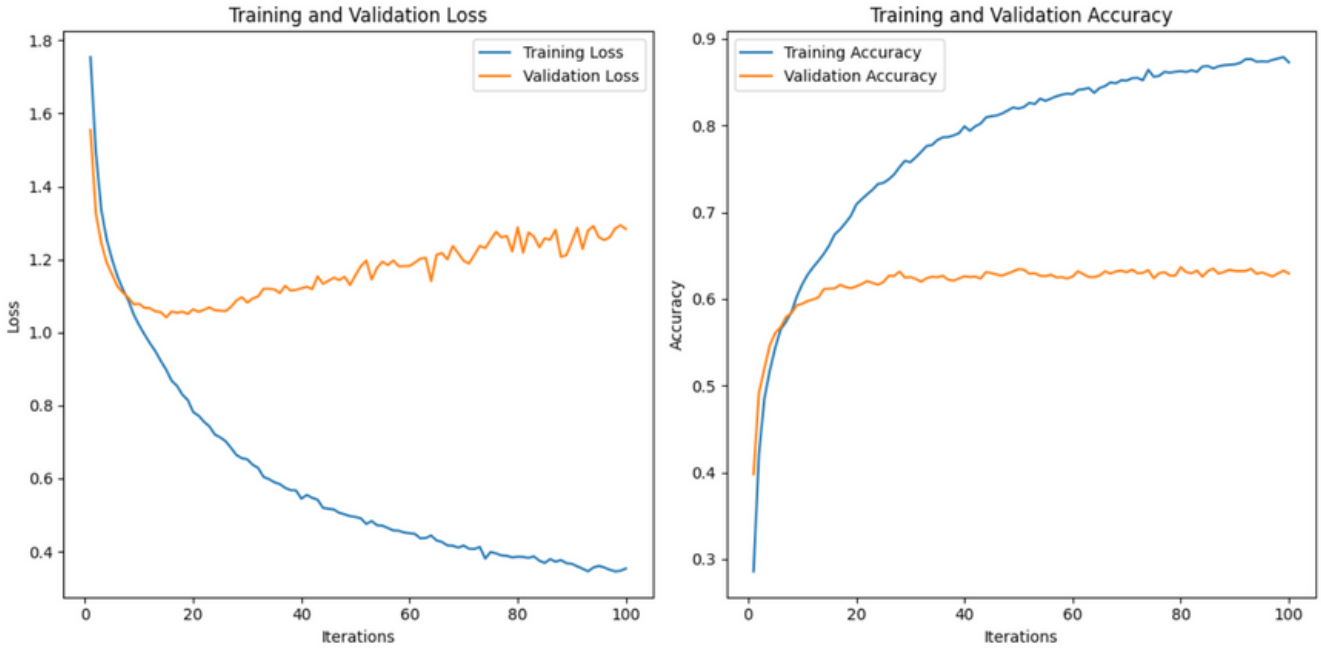
The hyperparameter tuning process involved exploring different learning rates, batch sizes, and dropout rates to find combinations that improved the model's performance. Overall, the tuning process aimed to find hyperparameters that resulted in lower validation loss and higher validation accuracy.

While some variations in hyperparameters led to improved performance, such as reducing the batch size and applying data augmentation techniques, other variations did not yield significant improvements. It's important to note that the tuning process involved iterating and testing different hyperparameter combinations to find the best configuration for the model.

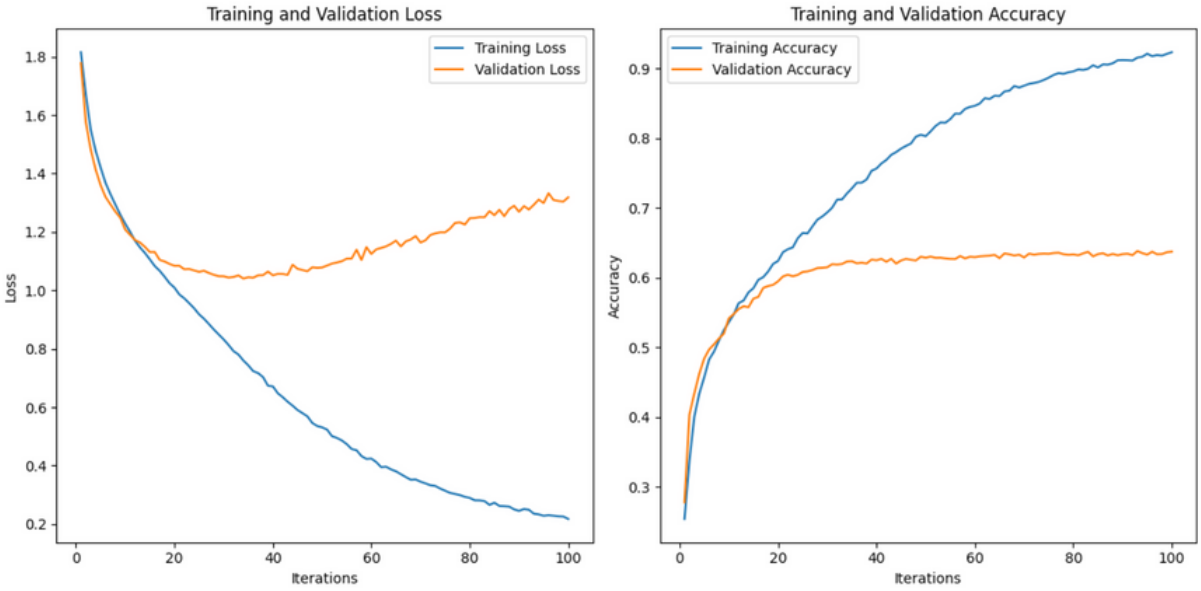
Our method trains a CNN model for emotion detection using the FER 2013 dataset. The model is trained by iterating over the specified number of epochs and updating the weights based on the calculated loss.

Version 1

Version 2

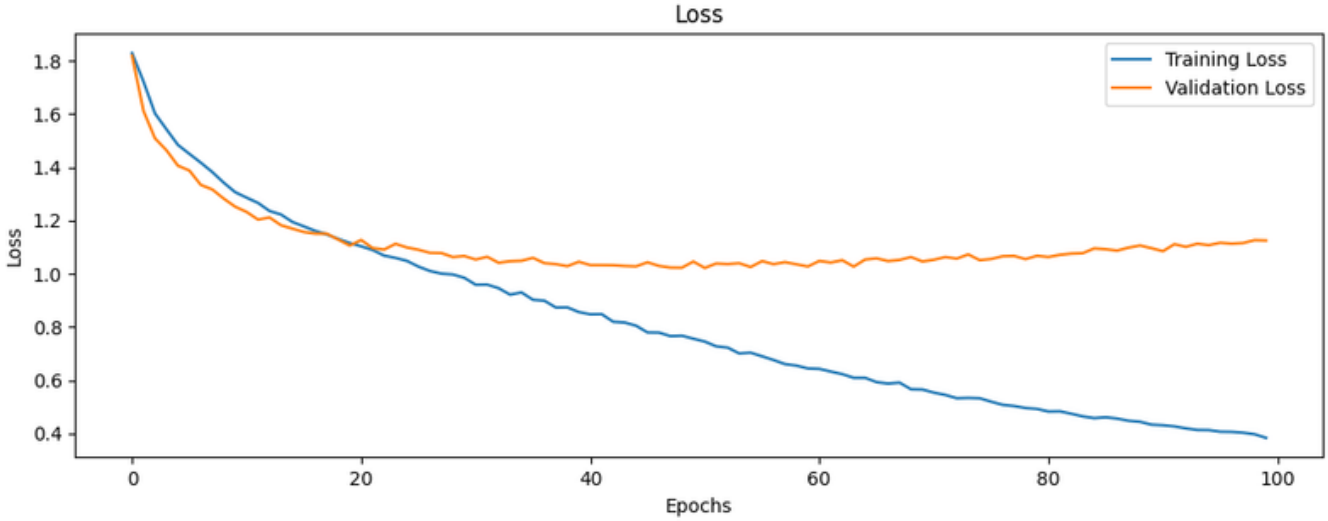


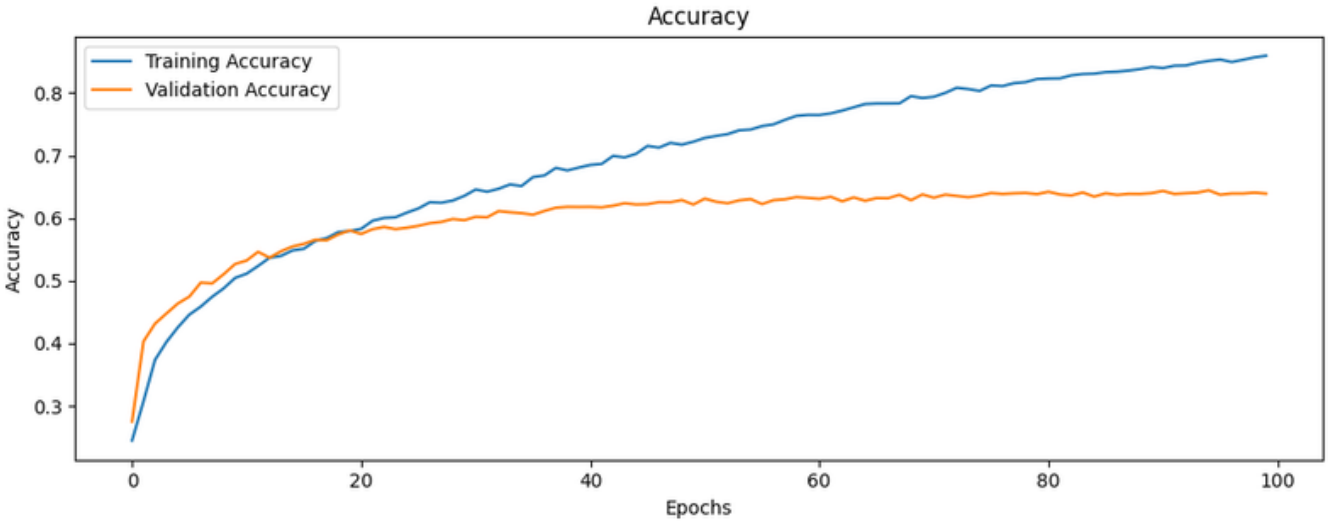
Version 3



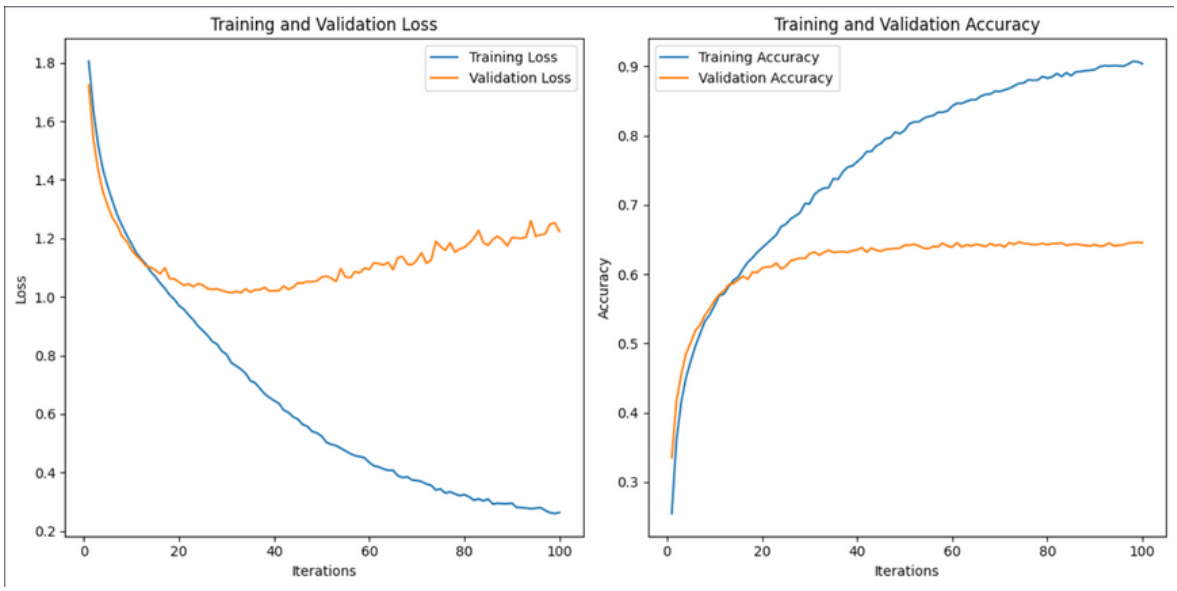
Version 4

Version 7

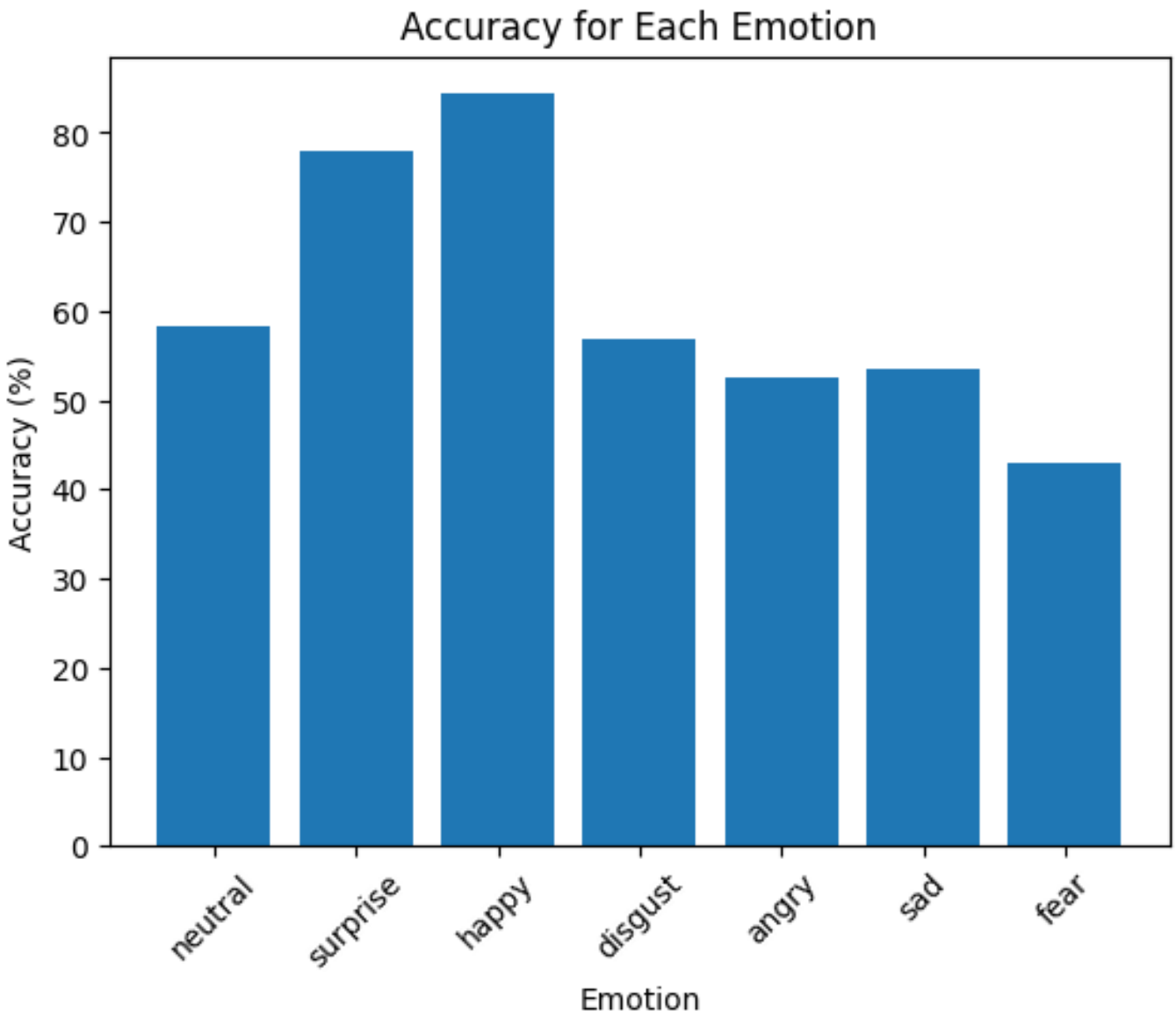




Version 5



ACCURACY OF EMOTIONS

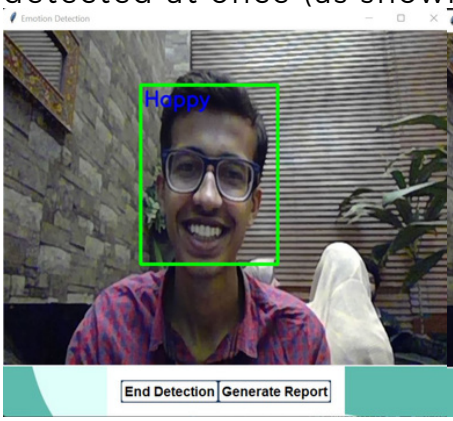


The higher accuracy for the "Happy" emotion category compared to the other emotions:

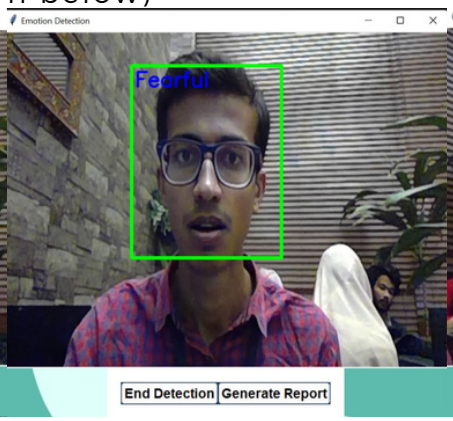
1. Data Representation: The dataset used for training and validation might have a well-represented and diverse range of "Happy" emotion examples. The availability of a sufficient number of accurately labeled and diverse "Happy" samples allows the model to learn and generalize well for this specific emotion.
2. Distinctive Features: The "Happy" emotion often exhibits distinct facial expressions and features, such as smiling, bright eyes, or raised cheeks. These distinctive features can potentially be more easily captured by the model, enabling it to differentiate and classify "Happy" samples accurately.
3. Model Architecture: The architecture of the model itself may be well-suited for capturing the visual cues and patterns associated with the "Happy" emotion.

REAL TIME EMOTION DETECTION

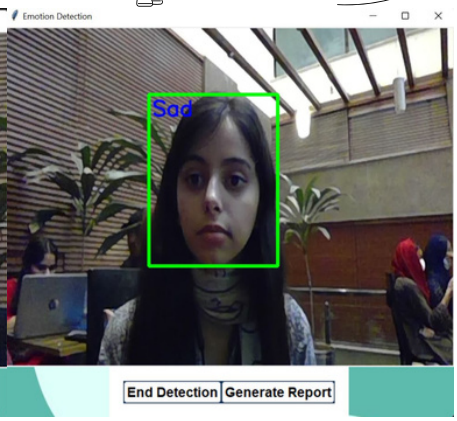
Demonstration of real-time detection done using the webcam feed.
The rectangle/square represents the detection, and multiple can be detected at once (as shown below)



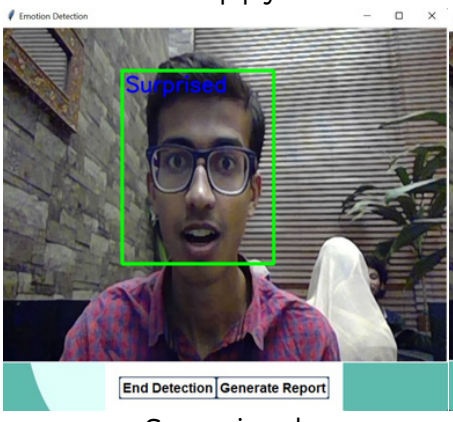
Happy



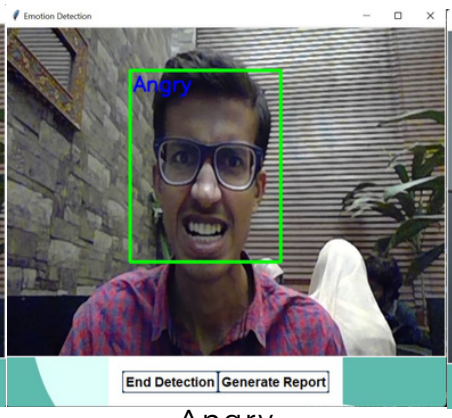
Fearful



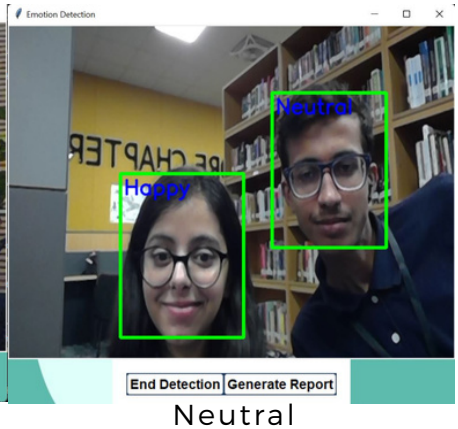
Sad



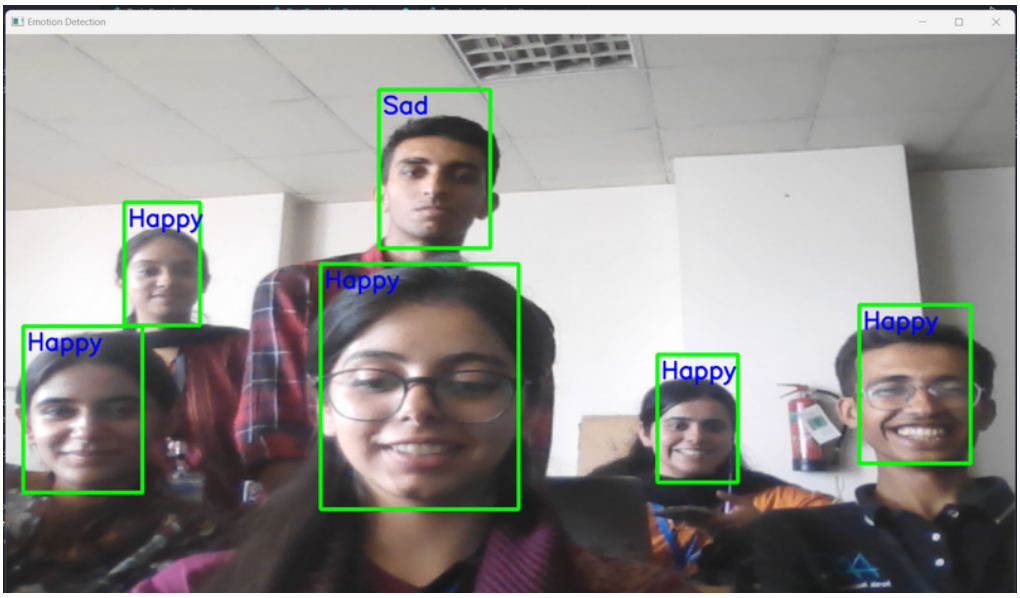
Surprised



Angry

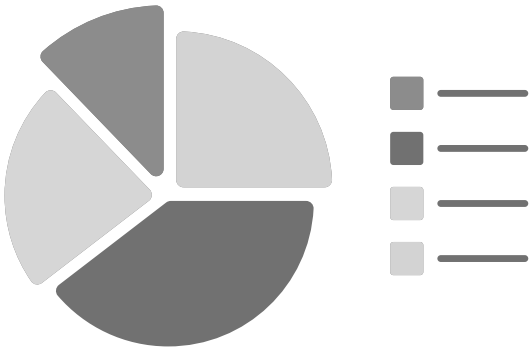


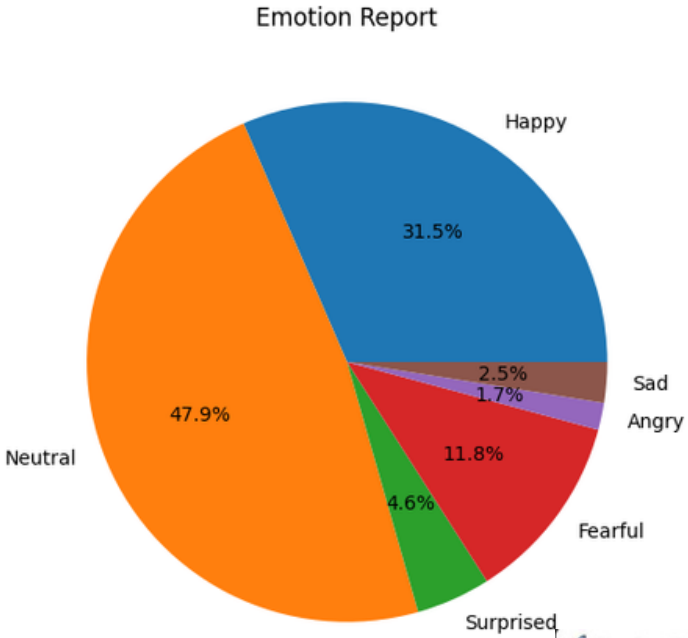
Neutral

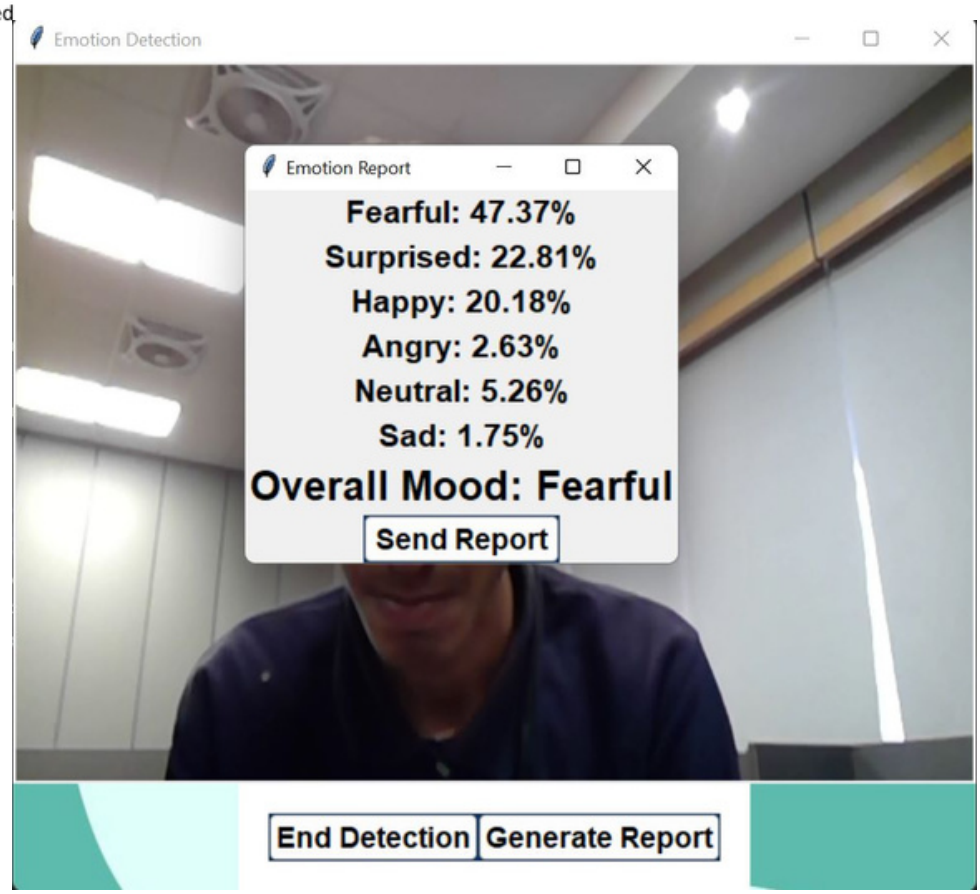


EMOTION REPORT

At the end of real time emotion detection, the percentages of emotions felt are shown using the dictionaries and lists to store all emotions and keep a count to distribute the percentage.







IMPLEMENTATION

TRAINING

TrainEmotionDetector.ipynb

Imports

```
import cv2
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
```

- Open CV imported for image processing (cv2).
- The sequential model was imported for the building of a neural network from Keras models.
- Conv2D, MaxPooling 2D, Dense, Dropout, and Flatten layers are imported to use between input and out of the model.
- Adam optimizer imported for the model optimization.
- Image data generator imported from Keras preprocessing of images.

Data Normalization

```
train_data_gen = ImageDataGenerator(rescale=1./255)
validation_data_gen = ImageDataGenerator(rescale=1./255)
```

this is to augment the dataset in real time while our model is still in the training stage. It works by creating multiple transformed versions of the images in the dataset, helping to expose the model to more diverse data and so reduces overfitting.
The parameter **rescale=1./255** is used to normalize pixel values in images. In an image, pixel intensity values usually range from 0 (black) to 255 (white). When training deep learning models, it's a common practice to normalize these pixel values so that they fall in the range [0, 1]. This helps the model learn more effectively because smaller input values are generally easier for the model to process.
Thus, initializing image data generators for training and validation datasets will scale image pixel intensities from the range [0, 255] to the range [0, 1].

Batches Generated

```
train_generator = train_data_gen.flow_from_directory(
    'C:\\Users\\farwa\\.virtualenvs\\.virtualenvs\\EmotionCNN\\train',
    target_size=(48, 48),
    batch_size=32,
    color_mode="grayscale",
    class_mode='categorical'
)

validation_generator = validation_data_gen.flow_from_directory(
    'test',
    target_size=(48, 48),
    batch_size=32,
    color_mode="grayscale",
    class_mode='categorical'
)
```

- The flow_from_directory method is used to generate batches of data from image files in a directory.
- It automatically labels the data based on the directory structure.
- The method takes in the directory path as a parameter and loads all the images from that directory.
- The images are preprocessed and resized to a specified target size.
- It can generate different types of labels, including categorical labels.
- The method returns a generator object that can be used to iterate over batches of training or validation data during the model training process.
- They're created using the flow_from_directory method for the training and validation data, respectively.
- The generators are configured with the appropriate parameters, such as the target size, batch size, color mode, and class mode.
- The train_generator and validation_generator are used during the model training process to provide batches of data to the model for each training epoch and validation step.
- The flow_from_directory method simplifies the process of loading, preprocessing, and labeling image data from directories, making it easier to train models on large datasets.

Architecture of emotion model

```
emotion_model = Sequential()

emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(512, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))

cv2.ocl.setUseOpenCL(False)
```

- OpenCL is a framework that allows for parallel computing across different devices, such as GPUs (Graphics Processing Units) and CPUs (Central Processing Units). It can provide performance improvements in certain scenarios.
- Done to ensure compatibility or avoid potential issues arising from using OpenCL on the specific hardware or software configuration.

Layer Number	Layer Type	Notes
1	Convolutional	32 filters, each using a 3x3 kernel, with an activation function of ReLu. The input shape is 48x48 for the size of the images and in one dimension.
2	Convolutional	64 filters, 3x3 kernel, and uses activation function ReLu again
3	Convolutional	64 filters, 3x3 kernel, and uses activation function ReLu again
4	Max-pooling	Downsamples by taking the maximum value within a 2x2 window
5	Dropout	This layer that randomly sets 25% of the input units to 0 during its training, this helps to reduce overfitting
6	Convolutional	128 filters, and a 3x3 kernel with ReLu
7	Convolutional	128 filters, and a 3x3 kernel with ReLu
8	Max-pooling	pool size of 2x2
9	Convolutional	4th convolutional layer has 128 filters, 3x3 kernel, and uses ReLu activation function
10	Max-pooling	size of 2x2
11	Dropout	25% of inputs set to 0 for training
12	Flatten	This flattens the output of the 11th layer to a 1D vector so it can be ready for fully connected layers

Layer Number	Layer Type	Notes
13	Dense	This is the first fully connected layer which has 1024 units and ReLu activation function. It performs a product of all neurons weights and activations from previous layers. This leads to learn non-linear relationships between input features
14	Dropout	Sets 50% input units to 0
15	Dense	512 units with activation function ReLu
16	Dropout	50% input units to 0
17	Dense	This is the last layer, fully connected with 7 units for the 7 emotion. Using softmax as the activation function to output probabilities for each class

Compilation Before Training

```
emotion_model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001, decay=1e-6), metrics=['accuracy'])
```

- loss='categorical_crossentropy': This specifies the loss function used for training the model. Categorical cross-entropy is used for multi-class classification problems, where the target variable is categorical and each class is mutually exclusive.
- optimizer=Adam(lr=0.0001, decay=1e-6): This defines the optimizer used to update the model's weights during training. The Adam optimizer is a popular choice for deep learning models. Here, it is configured with a learning rate (lr) of 0.0001 and a decay rate of 1e-6, which gradually reduces the learning rate over time.
- metrics=['accuracy']: This specifies the evaluation metric used to monitor the model's performance during training. Here we used an accuracy metric to measure how well the model predicts the correct emotion category.
- By calling compile() on the emotion_model, the model is configured and ready for training. The loss function, optimizer, and evaluation metric are set according to the specified parameters.

Training

```
emotion_model_info = emotion_model.fit(
    train_generator,
    steps_per_epoch=28709 // 32,
    epochs=100,
    validation_data=validation_generator,
    validation_steps=7178 // 32
)
```

- emotion_model.fit_generator(): This method is used to train the model using data from a generator. It takes the training generator (train_generator) as the input.
- steps_per_epoch=28709 // 32: This parameter specifies the number of steps (batches) to iterate over the training data generator in each epoch. In this case, it is calculated by dividing the total number of training samples (28709) by the batch size (32).
- epochs=100: This parameter defines the number of training epochs, which is the number of times the model will iterate over the entire training dataset.
- validation_data=validation_generator: This parameter specifies the validation generator (validation_generator) to be used for evaluating the model's performance during training.
- validation_steps=6781 // 32: This parameter determines the number of steps (batches) to iterate over the validation data generator in each epoch. It is calculated by dividing the total number of validation samples (6781) by the batch size (32).
- The training process will update the model's weights and optimize them based on the specified loss function and optimizer. The progress and performance metrics, such as loss and accuracy, will be recorded and stored in the emotion_model_info variable.

Training

```
# save model structure in JSON file
model_json = emotion_model.to_json()
with open("emotion_model_redu_v2.json", "w") as json_file:
    json_file.write(model_json)

# save trained model weights in .h5 file
emotion_model.save_weights('emotion_model_redu_v2.h5')
```

By saving both the model structure and weights, you can later load and use the trained model for testingor further training without having to retrain the model from scratch.

IMPLEMENTATION

TESTING

TestEmotionDetector.ipynb

Imports

```
import cv2
import numpy as np
from tkinter import Tk, Button, Toplevel, Label, messagebox, Frame, filedialog, Entry
from PIL import Image, ImageTk
from keras.models import model_from_json
from tkinter import ttk
```

- Import specific classes and functions from the tkinter module for GUI creation.
- Import Image and ImageTk modules from PIL for image processing and Tkinter integration.
- Import model_from_json function from keras.models for loading model architecture from JSON.
- Import ttk module from tkinter for themed widget styling.

Loading Model

```
# save model structure in JSON file
model_json = emotion_model.to_json()  # Convert the model s
with open("emotion_model_updated.json", "w") as json_file:
    json_file.write(model_json)  # Write the JSON structure
```

- Open the JSON file containing the model architecture in read mode and assign it to the json_file variable.
- Read the content of the opened JSON file and assign it to the loaded_model_json variable.
- Close the opened JSON file.
- Load the model architecture from the JSON content stored in the loaded_model_json variable using model_from_json function from Keras, and assign it to the emotion_model variable.

Creating Dictionary

```
# Load weights into the model
emotion_model.load_weights("model/emotion_model_v4.h5")

emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6: "Surprised"}
```

- Load the weights of a pre-trained emotion detection model from a file named "emotion_model_v4.h5" located in the "model" directory.
- Create a dictionary called emotion_dict that maps integer labels to corresponding emotion names.

end_detection function

```
def end_detection():
    window2.destroy()
    window.deiconify()
```

- The function end_detection() closes window2 (a GUI window) and brings back the previously hidden window to the foreground.

start_detection function

```
def start_detection():
    global cap, window2, label_img, stop_camera, emotion_report

    window.withdraw()  # Hide the first window

    cap = cv2.VideoCapture(0)
    stop_camera = False
    emotion_report = []  # Reset emotion report

    window2 = Toplevel()
    window2.title("Emotion Detection")

    background_image = Image.open("background_image.jpg")
    background_image = background_image.resize((1400, 1000), Image.LANCZOS)
    background_photo = ImageTk.PhotoImage(background_image)

    label_background = Label(window2, image=background_photo)
    label_background.place(x=0, y=0, relwidth=1, relheight=1)
    label_background.image = background_photo

    button_frame = Frame(window2, bg='white', padx=20, pady=20)
    button_frame.pack(side='bottom')
```

- The function start_detection() initializes variables, hides the main window, opens the camera, creates a new window with a background image, and sets up a frame for buttons.

stop_detection

```
def stop_detection():
    global stop_camera
    stop_camera = True
    window2.destroy()
    window.deiconify()
```

- The function stop_detection() sets the stop_camera flag to True, closes window2 (a GUI window), and brings back the previously hidden window to the foreground.

generate_report function

```
def generate_report():
    global stop_camera, emotion_report
    stop_camera = True

    if not emotion_report:
        messagebox.showerror("Error", "No emotions detected. Please try again.")
        return

    emotion_count = {}
    total_emotions = len(emotion_report)

    for emotion in emotion_report:
        if emotion in emotion_count:
            emotion_count[emotion] += 1
        else:
            emotion_count[emotion] = 1

    emotion_percentage = {}
    for emotion, count in emotion_count.items():
        percentage = (count / total_emotions) * 100
        emotion_percentage[emotion] = percentage

    overall_mood = max(emotion_percentage, key=emotion_percentage.get)

    report_window = Toplevel()
    report_window.title("Emotion Report")

    # Set the background image for the report window
    background_image = Image.open("background_image.jpg")
    background_image = background_image.resize((1500, 1000), Image.LANCZOS)
    background_photo = ImageTk.PhotoImage(background_image)
    label_background = Label(report_window, image=background_photo)
    label_background.place(x=0, y=0, relwidth=1, relheight=1)
    label_background.image = background_photo  # Attach the image object to the label to prevent it from being garbage collected

    report_frame = Frame(report_window)
    report_frame.pack()

    for emotion, percentage in emotion_percentage.items():
        label = Label(report_frame, text=f"{emotion_dict[emotion]}: {percentage:.2f}%", font=("Arial", 16, "bold"))
        label.pack(anchor='center')

    label_overall_mood = Label(report_frame, text=f"Overall Mood: {emotion_dict[overall_mood]}", font=("Arial", 20, "bold"))
```

- The function generate_report() calculates the percentage of each emotion detected, determines the overall mood based on the highest percentage, and displays the emotion report in a new window with a background image and labels. It also sets the stop_camera flag to True to stop the emotion detection process.

send_report_to_therapist function

```
def send_report_to_therapist():
    report_window.destroy()
    details_window = Toplevel()
    details_window.title("Patient Details")

    # Set the background image for the details window
    background_image = Image.open("background_image.jpg")
    background_image = background_image.resize((1500, 1000), Image.LANCZOS)
    background_photo = ImageTk.PhotoImage(background_image)
    label_background = Label(details_window, image=background_photo)
    label_background.place(x=0, y=0, relwidth=1, relheight=1)
    label_background.image = background_photo

    details_frame = Frame(details_window)
    details_frame.pack()

    # Add labels and entry fields for patient details
    label_name = Label(details_frame, text="Name:", font=("Arial", 16, "bold"))
    label_name.pack()
    entry_name = Entry(details_frame, font=("Arial", 14))
    entry_name.pack()

    label_contact = Label(details_frame, text="Contact No:", font=("Arial", 16, "bold"))
    label_contact.pack()
    entry_contact = Entry(details_frame, font=("Arial", 14))
    entry_contact.pack()

    label_medical = Label(details_frame, text="Medical Issues:", font=("Arial", 16, "bold"))
    label_medical.pack()
    entry_medical = Entry(details_frame, font=("Arial", 14))
    entry_medical.pack()
```

- The function send_report_to_therapist() closes the report_window, creates a new window (details_window) for patient details, and adds labels and entry fields for the patient's name, contact number, and medical issues within a frame (details_frame).

send_report function

```
def send_report():
    name = entry_name.get().strip()
    contact = entry_contact.get().strip()
    medical_issues = entry_medical.get().strip()

    if not name or not contact or not medical_issues:
        messagebox.showerror("Error", "Please enter all details.")
        return

    messagebox.showinfo("Report Sent", "The report has been sent successfully!")
    details_window.destroy()
```

- The function send_report() retrieves the values entered in the entry fields for name, contact, and medical issues, checks if any of the fields are empty, shows an error message if any field is empty, shows a success message if all fields are filled, and closes the details_window.

Real-time Emotion Detection from Camera Feed

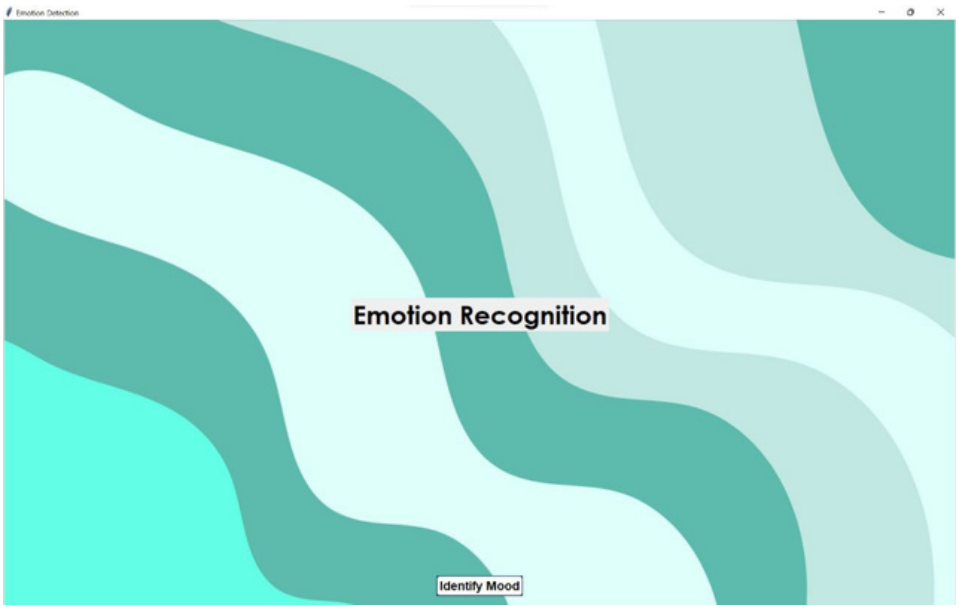
```
while not stop_camera:
    ret, frame = cap.read()
    frame = cv2.resize(frame, (640, 480))
    if not ret:
        break
    frame = cv2.flip(frame, 1)

    face_detector_front = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
    gray_frame_front = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    num_faces = face_detector_front.detectMultiScale(gray_frame_front, scaleFactor=1.3, minNeighbors=5)

    for (x, y, w, h) in num_faces:
        cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (0, 255, 0), 4)
        roi_gray_frame_front = gray_frame_front[y:y + h, x:x + w]
        cropped_img_front = np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame_front, (48, 48)), -1), 0)

        emotion_prediction_front = emotion_model.predict(cropped_img_front)
        maxindex_front = int(np.argmax(emotion_prediction_front))
        cv2.putText(frame, emotion_dict[maxindex_front], (x+5, y-20), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA)

        emotion_report.append(maxindex_front)

    img = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    img_tk = ImageTk.PhotoImage(image=img)
    label_img.configure(image=img_tk)
    label_img.image = img_tk

    window2.update()

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

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

- It continuously captures frames from the camera until the stop_camera flag is set to True. It performs face detection on each frame, draws rectangles around detected faces, extracts the region of interest (ROI) for each face, resizes it, and feeds it into an emotion detection model. The predicted emotion label is displayed on each detected face. The processed frame is converted to an image, displayed in a label widget (label_img), and appended to the emotion_report list.

RESULTS AND OBSERVATIONS

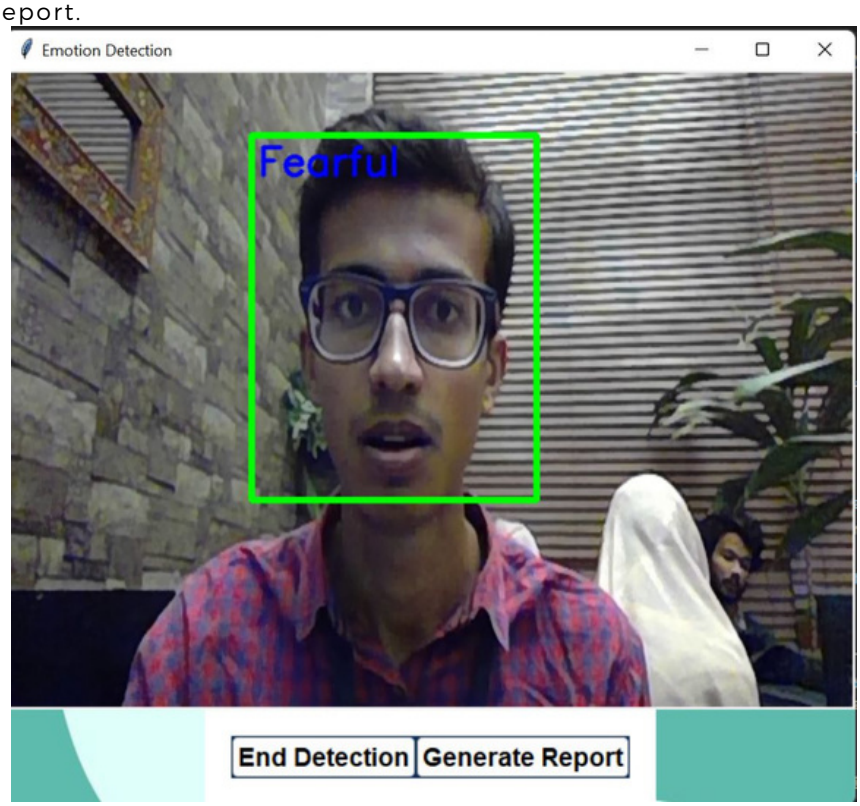
FINAL OUTPUTS

1: Start Detection



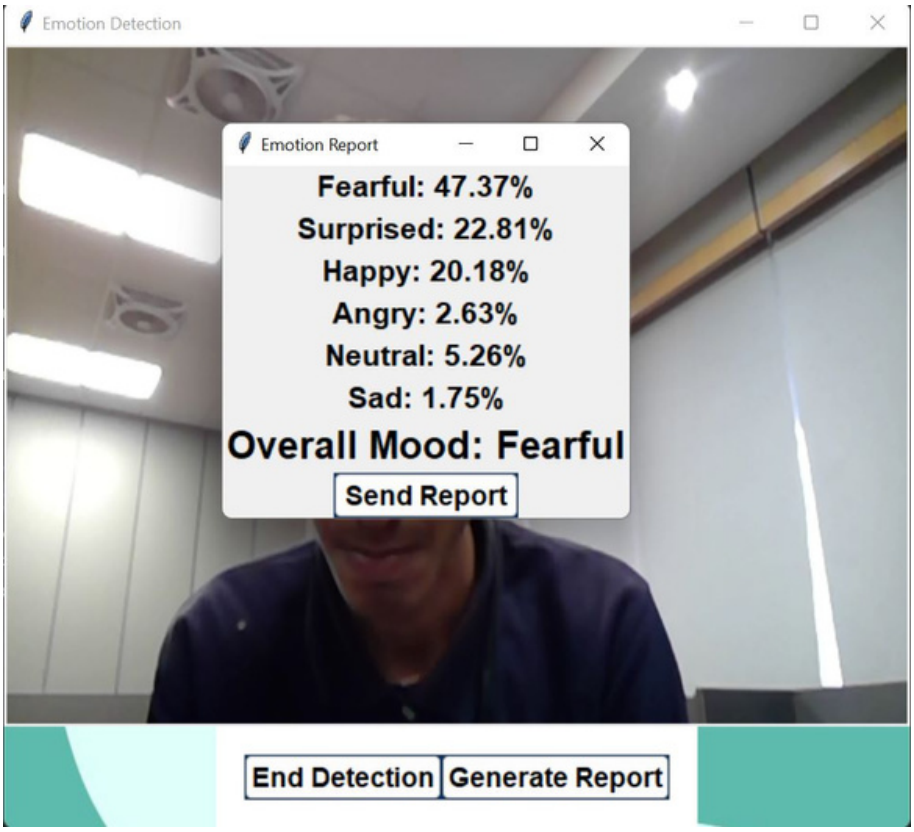
2: Camera On

The camera turns on and the emotion detection begins, while simultaneously storing the data to generate the report.



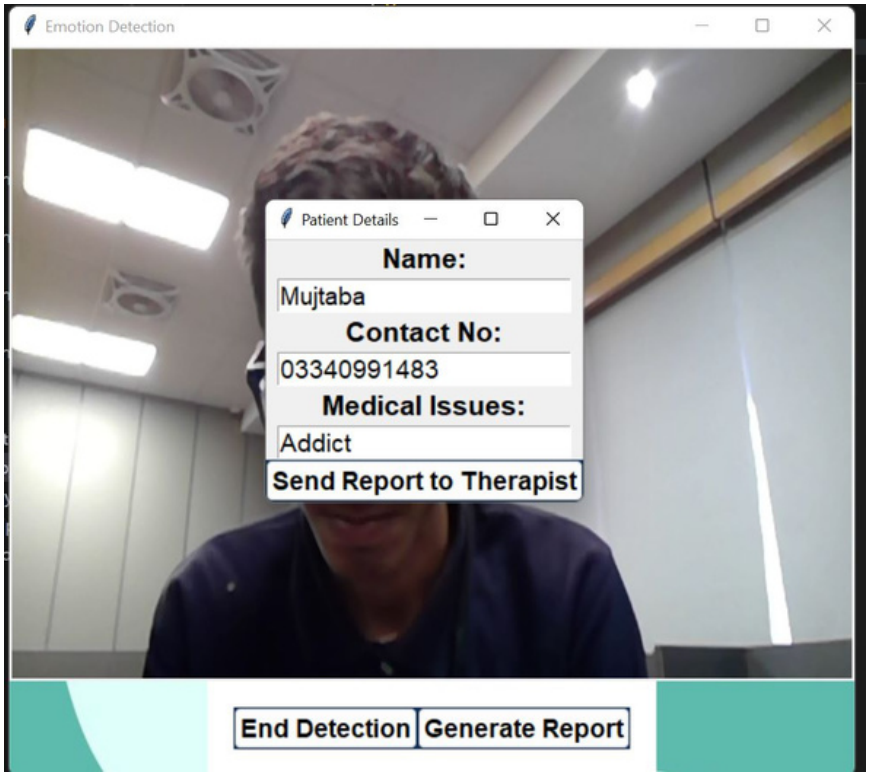
3: Generate Report

The Percentages of every emotion felt is generated and displayed.



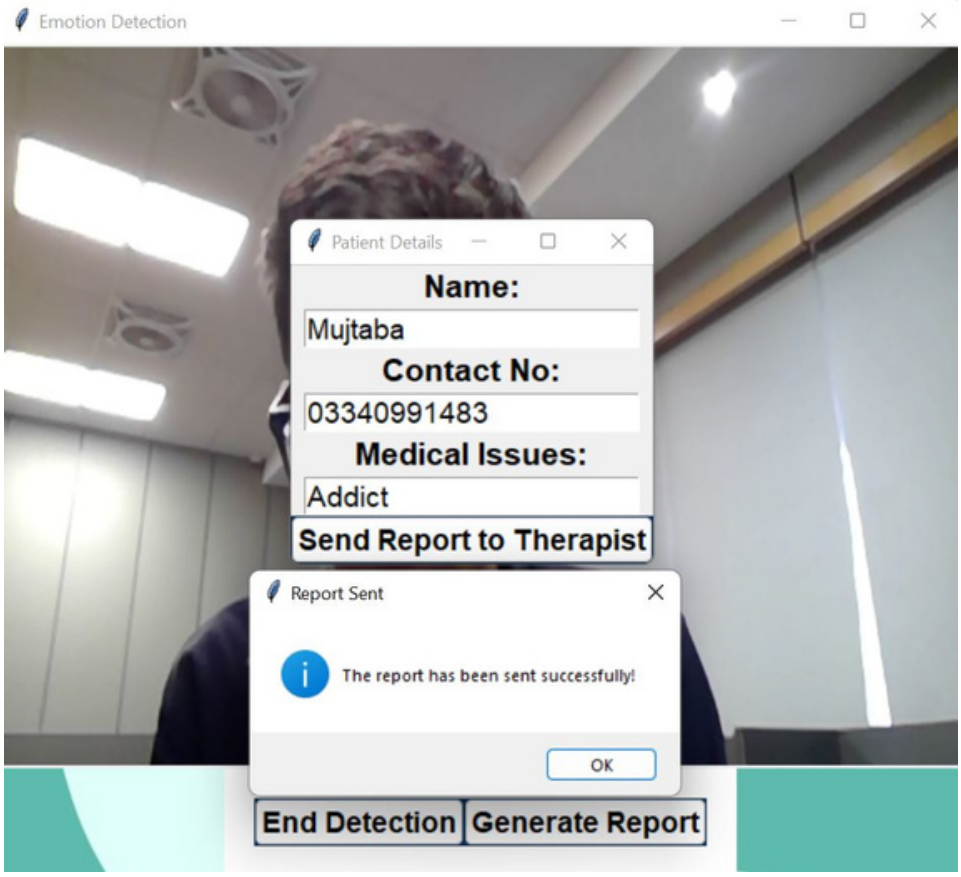
4: Patient Details

To send report the patients details are requested.



5: Send Report

Report is sent to the therapist.



LIMITATIONS & ALTERNATIVES

LIMITATIONS

- Dependence on Haar Cascade Classifier: The code uses a Haar cascade classifier for face detection, which may have limitations in detecting faces accurately in conditions such as low lighting or non-frontal faces.
- Limited to Facial Expressions: The system is designed specifically for facial expression analysis. It may not be suitable for detecting emotions in other types of images or for recognizing emotions based on other modalities like speech or text.
- Low accuracy (76.4%) for the emotion like “fear”
- Emotions may find it difficult to detect if the person is wearing any prop like a hat, cap or goggles.
- The dataset used has noisy data hence we were not able to achieve accuracy in the 90’ish range.
- Detected emotion doesn’t specify the actual mood of the person or how the person is feeling in reality, this is just a peripheral analysis and detection of human emotions.
- The dataset used is Fer2013, and access to other private datasets are not found easily.

ALTERNATIVES

- Deep Learning Architectures: Instead of using a simple CNN architecture, more advanced architectures like Recurrent Neural Networks (RNNs) consider the sequence of frames or images in a video or a series of facial images; or Transformers can be explored to capture long-range dependencies in emotion recognition.

However, we found CNN to be most suited as RNN may fail to retain the relevant information over long sequences, and Transformers are NLP, also involving a large number of parameters, making them computationally expensive to train and deploy

CONCLUSION

In conclusion, the computer vision emotion detector project successfully developed a deep learning model using Convolutional Neural Networks (CNNs) to recognize emotions from facial expressions. The model underwent hyperparameter tuning and achieved decent accuracy in emotion classification. However, there are limitations related to lighting, pose, and image quality. Future improvements could explore advanced techniques like RNNs or Transformers to capture temporal dependencies and enhance emotion recognition in sequential data.

Overall, the computer vision emotion detector semester project provided valuable insights into the application of deep learning techniques for emotion recognition from facial expressions. It served as a foundation for further research and development in the field of affective computing, with the potential to contribute to various domains, including human-computer interaction, psychology, and healthcare.

REFERENCES

- Ekman, P. (1992). An argument for basic emotions. Cognition and Emotion, 6(3-4), 169-200.

- Bartlett, M. S., Littlewort, G., Frank, M. G., Lainscsek, C., Fasel, I., & Movellan, J. (2005). Automatic recognition of facial actions in spontaneous expressions. Journal of Multimedia, 1(6), 22-35.

- Kotsia, I., & Pitas, I. (2008). Emotion recognition in the wild by exploiting points of interest. IEEE Transactions on Affective Computing, 2(2), 92-105.

- Li, X., Hu, S., Zhang, Z., Zhang, X., Zhang, Z., Zhang, L., & Li, L. (2017). Facial expression recognition based on deep learning: A comprehensive review. arXiv preprint arXiv:1704.08348.

- Dhall, A., Goecke, R., Joshi, J., Wagner, M., & Gedeon, T. (2012). Emotion recognition in the wild challenge 2012. In Proceedings of the 14th ACM International Conference on Multimodal Interaction (pp. 503-510).

- Liu, M., Zhang, S., Chen, L., Li, J., & Zhang, X. (2018). Deep learning for emotion recognition: A survey. Neurocomputing, 324, 3-13.

- Sariyanidi, E., Gunes, H., & Cavallaro, A. (2015). Automatic analysis of facial affect: A survey of registration, representation, and recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(6), 1113-1133.