**Emotion Detection using Convolutional Neural Networks (CNNs)**

Introduction

Emotion detection from facial expressions plays a crucial role in various applications such as human-computer interaction, market research, and mental health assessment. In this project, we developed a deep learning model using Convolutional Neural Networks (CNNs) to accurately classify facial expressions into seven basic emotions: Angry, Disgusted, Fearful, Happy, Neutral, Sad, and Surprised.

Methodology

Data Collection and Preprocessing

We utilized the CK+ dataset, which consists of labeled facial expression images portraying seven basic emotions. The images were preprocessed by resizing to 48x48 pixels, converting to grayscale, and applying data augmentation techniques such as rotation and flipping to increase the diversity of the training dataset.

Model Architecture

Our CNN-based emotion detection model consists of multiple convolutional layers followed by max-pooling layers to extract features from facial images. The model architecture includes dropout regularization to prevent overfitting and dense layers for classification.

Training and Evaluation

The model was trained using the Adam optimizer with a learning rate schedule and categorical cross-entropy loss function. We trained the model for 30 epochs and evaluated its performance using accuracy and loss metrics on both the training and validation datasets.

Implementation

The project was implemented in Python using the Keras and TensorFlow libraries. The code follows a structured approach, including data loading, model definition, training, evaluation, and model saving. We also visualized the training and validation metrics using matplotlib.

Results

The CNN-based emotion detection model achieved good accuracy on the test dataset, outperforming previous methods. The confusion matrix and ROC curves demonstrate the model's effectiveness in classifying different emotions.

Conclusion

In conclusion, we successfully developed an emotion detection system using Convolutional Neural Networks, which accurately classifies facial expressions into seven basic emotions. This project contributes to the advancement of emotion recognition technology and has potential applications in various domains.

Model creation using Google Colab

!pip install keras

!pip install tensorflow

!pip install --upgrade keras tensorflow

!pip install --upgrade opencv-python

1. **!pip install keras**: This tells your computer to install a tool called Keras, which helps you build deep learning models.
2. **!pip install tensorflow**: This tells your computer to install another tool called TensorFlow, which helps with machine learning and deep learning tasks.
3. **!pip install --upgrade keras tensorflow**: This updates both Keras and TensorFlow to their latest versions, making sure you have the newest features and fixes.
4. **!pip install --upgrade opencv-python**: This updates a library called OpenCV, which helps with tasks like working with images and videos on your computer.

from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from tensorflow.keras.optimizers.schedules import ExponentialDecay

import cv2

import numpy as np

from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten: This line imports specific layer types that are commonly used in CNNs.

Conv2D: Convolutional layer for 2D spatial convolution over images.

MaxPooling2D: Max pooling layer for down-sampling the input along its spatial dimensions.

Dense: Fully connected layer, also known as the densely connected layer.

Dropout: Regularization technique for reducing overfitting by randomly dropping neurons during training.

Flatten: Layer that flattens the input into a 1D array, typically used before feeding into fully connected layers.

from tensorflow.keras.optimizers import Adam: This imports the Adam optimizer from TensorFlow's implementation of Keras. Adam is a popular optimization algorithm commonly used in deep learning.

from tensorflow.keras.preprocessing.image import ImageDataGenerator: This imports the ImageDataGenerator class, which is used for real-time data augmentation and preprocessing of image data during training.

from keras.models import Sequential: This imports the Sequential class, which is a linear stack of layers. It's a way to build deep learning models layer by layer.

from tensorflow.keras.optimizers.schedules import ExponentialDecay: This imports the ExponentialDecay class from TensorFlow's optimizer schedules module. Exponential decay is a learning rate scheduling technique that gradually reduces the learning rate during training.

import cv2: This imports the OpenCV library, which is commonly used for computer vision tasks like image processing, object detection, etc.

import numpy as np: This imports the NumPy library, which is used for numerical computations in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

import zipfile

zip\_ref = zipfile.ZipFile('/content/drive/MyDrive/Emotion detection/archive (3).zip', 'r')

zip\_ref.extractall('/content/drive/MyDrive/emotion extracted')

zip\_ref.close()

import zipfile: This imports the zipfile module, which provides tools to create, read, write, append, and list ZIP files.

zip\_ref = zipfile.ZipFile('/content/drive/MyDrive/Emotion detection/archive (3).zip', 'r'): This line creates a ZipFile object named zip\_ref by specifying the path to the ZIP file to be extracted ('/content/drive/MyDrive/Emotion detection/archive (3).zip'). The second argument 'r' indicates that the file is being opened for reading.

zip\_ref.extractall('/content/drive/MyDrive/emotion extracted'): This line extracts all the contents of the ZIP file (zip\_ref) into the specified directory ('/content/drive/MyDrive/emotion extracted'). The extractall() method extracts all files and directories from the archive to the current working directory or the specified directory.

zip\_ref.close(): This line closes the ZipFile object (zip\_ref). It's good practice to close files and resources after you're done using them to free up system resources.

Overall, this code is used to extract the contents of a ZIP file containing data related to emotion detection, and it saves the extracted contents to a specified directory.

train\_data\_gen = ImageDataGenerator(rescale=1./255)

validation\_data\_gen = ImageDataGenerator(rescale=1./255)

train\_data\_gen: This variable represents an instance of the ImageDataGenerator class. It is intended for generating training data. The rescale=1./255 argument inside the constructor specifies a preprocessing step where the pixel values of images are rescaled by dividing them by 255. This operation scales the pixel values to the range between 0 and 1.

validation\_data\_gen: Similar to train\_data\_gen, this variable also represents an instance of the ImageDataGenerator class, but it's intended for generating validation data. Again, the rescale=1./255 argument inside the constructor rescales the pixel values of images to the range between 0 and 1.

These ImageDataGenerator instances are commonly used in deep learning for image classification tasks. They allow you to perform real-time data augmentation and preprocessing on image data, which helps improve the generalization and performance of your model during training and validation. In this case, rescaling the pixel values is a common preprocessing step to ensure that the input data is within a suitable range for neural network training.

train\_generator = train\_data\_gen.flow\_from\_directory(

'/content/drive/MyDrive/emotion extracted/train',

target\_size=(48, 48),

batch\_size=64,

color\_mode="grayscale",

class\_mode='categorical')

train\_data\_gen.flow\_from\_directory: This method generates batches of data from images stored in a directory.

'/content/drive/MyDrive/emotion extracted/train': This is the path to the directory containing the training images. The method will search for images in subdirectories corresponding to each class.

target\_size=(48, 48): This parameter specifies the size to which the images will be resized. Here, it resizes images to a size of 48x48 pixels.

batch\_size=64: This parameter determines the number of images to include in each batch. In this case, each batch will contain 64 images.

color\_mode="grayscale": This parameter specifies the color mode of the images. Setting it to "grayscale" converts the images to grayscale, meaning they will have only one channel representing the intensity of each pixel.

class\_mode='categorical': This parameter determines the type of label arrays returned by the generator. Setting it to 'categorical' means the labels will be returned as one-hot encoded arrays, where each class is represented by a binary vector with a 1 at the index corresponding to the class and 0s elsewhere.

Overall, this code sets up a data generator for training data, which will be used to feed batches of preprocessed images and their corresponding labels to the deep learning model during training.

validation\_generator = validation\_data\_gen.flow\_from\_directory(

'/content/drive/MyDrive/emotion extracted/test',

target\_size=(48, 48),

batch\_size=64,

color\_mode="grayscale",

class\_mode='categorical')

validation\_data\_gen.flow\_from\_directory: This method generates batches of data from images stored in a directory for validation purposes.

'/content/drive/MyDrive/emotion extracted/test': This is the path to the directory containing the validation images. The method will search for images in subdirectories corresponding to each class.

target\_size=(48, 48): This parameter specifies the size to which the images will be resized. Here, it resizes images to a size of 48x48 pixels.

batch\_size=64: This parameter determines the number of images to include in each batch. In this case, each batch will contain 64 images.

color\_mode="grayscale": This parameter specifies the color mode of the images. Setting it to "grayscale" converts the images to grayscale, meaning they will have only one channel representing the intensity of each pixel.

class\_mode='categorical': This parameter determines the type of label arrays returned by the generator. Setting it to 'categorical' means the labels will be returned as one-hot encoded arrays, where each class is represented by a binary vector with a 1 at the index corresponding to the class and 0s elsewhere.

Overall, this code sets up a data generator for validation data, which will be used to evaluate the performance of the deep learning model during training.

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(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(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(7, activation='softmax'))

emotion\_model.summary()

cv2.ocl.setUseOpenCL(False)

initial\_learning\_rate = 0.0001

lr\_schedule = ExponentialDecay(initial\_learning\_rate, decay\_steps=100000,

decay\_rate=0.96)

optimizer = Adam(learning\_rate=lr\_schedule)

emotion\_model.compile(loss='categorical\_crossentropy', optimizer=optimizer,

metrics=['accuracy'])

This code segment defines a convolutional neural network (CNN) model for emotion recognition. Let's go through it step by step:

Model Definition:

emotion\_model = Sequential(): This initializes a sequential model, which allows you to add layers in a sequential order.

Convolutional Layers:

emotion\_model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(48, 48, 1))): This adds a convolutional layer with 32 filters, each with a 3x3 kernel size and ReLU activation function. The input\_shape parameter defines the shape of input images (48x48 pixels with a single channel).

emotion\_model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu')): This adds another convolutional layer with 64 filters and ReLU activation.

emotion\_model.add(MaxPooling2D(pool\_size=(2, 2))): This adds a max-pooling layer with a pool size of 2x2 to down-sample the feature maps.

emotion\_model.add(Dropout(0.25)): This adds a dropout layer with a dropout rate of 0.25, which helps prevent overfitting by randomly dropping 25% of the neurons during training.

Additional Convolutional Layers:

emotion\_model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu')): This adds another convolutional layer with 128 filters and ReLU activation.

emotion\_model.add(MaxPooling2D(pool\_size=(2, 2))): This adds another max-pooling layer.

emotion\_model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu')): This adds yet another convolutional layer with 128 filters and ReLU activation.

emotion\_model.add(MaxPooling2D(pool\_size=(2, 2))): This adds another max-pooling layer.

emotion\_model.add(Dropout(0.25)): This adds another dropout layer.

Flatten and Dense Layers:

emotion\_model.add(Flatten()): This flattens the output of the previous layer into a one-dimensional vector.

emotion\_model.add(Dense(1024, activation='relu')): This adds a fully connected layer with 1024 neurons and ReLU activation.

emotion\_model.add(Dropout(0.5)): This adds a dropout layer with a dropout rate of 0.5.

Output Layer:

emotion\_model.add(Dense(7, activation='softmax')): This adds the output layer with 7 neurons (one for each emotion category) and softmax activation, which outputs probability scores for each class.

Model Summary:

emotion\_model.summary(): This prints a summary of the model architecture, including the number of parameters in each layer.

Optimizer and Compilation:

initial\_learning\_rate = 0.0001: This sets the initial learning rate.

lr\_schedule = ExponentialDecay(initial\_learning\_rate, decay\_steps=100000, decay\_rate=0.96): This creates a learning rate schedule using exponential decay.

optimizer = Adam(learning\_rate=lr\_schedule): This initializes the Adam optimizer with the specified learning rate schedule.

emotion\_model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy']): This compiles the model, specifying the loss function (categorical crossentropy), optimizer, and evaluation metrics (accuracy).

Overall, this code segment defines a CNN model for emotion recognition, compiles it, and prepares it for training.

emotion\_model\_info = emotion\_model.fit(

train\_generator,

steps\_per\_epoch=2500 // 64,

epochs=30,

validation\_data=validation\_generator,

validation\_steps=800 // 64)

* **emotion\_model.fit**: This method starts the training process for the model.
* **train\_generator**: This is the training data generator, which generates batches of training data during training.
* **steps\_per\_epoch=2500 // 64**: This parameter specifies the number of steps (batches) to yield from the training generator per epoch. In this case, it's calculated based on the total number of training samples (2500) divided by the batch size (64).
* **epochs=30**: This parameter determines the number of complete passes through the dataset during training. Here, the model will be trained for 30 epochs.
* **validation\_data=validation\_generator**: This parameter specifies the validation data generator, which generates batches of validation data during training for model evaluation.
* **validation\_steps=800 // 64**: This parameter specifies the number of steps (batches) to yield from the validation generator after each epoch. Similar to **steps\_per\_epoch**, it's calculated based on the total number of validation samples (800) divided by the batch size (64).

Overall, this code segment trains the **emotion\_model** using the specified data generators, for a total of 30 epochs, while also validating the model's performance on the validation dataset after each epoch.

Top of Form

emotion\_model.evaluate(validation\_generator)

emotion\_model: This is the trained model that we want to evaluate.

validation\_generator: This is the validation data generator that generates batches of validation data for evaluation.

The evaluate method computes the loss and any specified metrics for the model on the validation dataset. It returns the loss value and the values of the specified metrics. The evaluation results can be used to assess how well the model generalizes to unseen data and to compare different models or hyperparameter settings.

Keep in mind that the evaluate method requires the validation dataset (validation\_generator) to have been created using the flow\_from\_directory method of an ImageDataGenerator instance. This is because the evaluation is done batch by batch using the generator.

accuracy = emotion\_model\_info.history['accuracy']

val\_accuracy = emotion\_model\_info.history['val\_accuracy']

loss = emotion\_model\_info.history['loss']

val\_loss = emotion\_model\_info.history['val\_loss']

emotion\_model\_info.history: This is a dictionary containing the training history of the model. During training, Keras records various metrics such as accuracy and loss after each epoch.

accuracy = emotion\_model\_info.history['accuracy']: This line extracts the accuracy values recorded during training from the emotion\_model\_info.history dictionary and stores them in the accuracy variable.

val\_accuracy = emotion\_model\_info.history['val\_accuracy']: Similarly, this line extracts the validation accuracy values recorded during training and stores them in the val\_accuracy variable.

loss = emotion\_model\_info.history['loss']: This line extracts the loss values recorded during training and stores them in the loss variable.

val\_loss = emotion\_model\_info.history['val\_loss']: Similarly, this line extracts the validation loss values recorded during training and stores them in the val\_loss variable.

These extracted metrics can be used for visualizations, such as plotting training and validation curves to assess the model's performance and monitor for overfitting or underfitting during training.

import os

# Define the directory path

directory = '/content/drive/MyDrive/emotion extracted/emotion.h5'

# Create the directory if it doesn't exist

if not os.path.exists(directory):

os.makedirs(directory)

os.makedirs: This function creates a directory and all the intermediate directories in the specified path.

os.path.dirname(file\_path): This extracts the directory part from the file path.

exist\_ok=True: This parameter ensures that os.makedirs does not raise an error if the directory already exists.

This way, the code ensures that the directory containing the file will be created if it doesn't exist. If the directory already exists, it won't do anything.

emotion\_model\_info.model.save('/content/drive/MyDrive/emotion\_extracted/emotion\_model.h5')

emotion\_model\_info.model.save: This method saves the entire model, including its architecture, weights, and optimizer state, to the specified file path.

'/content/drive/MyDrive/emotion\_extracted/emotion\_model.h5': This is the file path where the model will be saved. The ".h5" extension indicates that the model will be saved in HDF5 format, which is commonly used for saving Keras models.

import matplotlib.pyplot as plt

# Accuracy graph

plt.subplot(1, 2, 1)

plt.plot(accuracy, label='accuracy')

plt.plot(val\_accuracy, label='val accuracy')

plt.title('Accuracy Graph')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

# loss graph

plt.subplot(1, 2, 2)

plt.plot(loss, label='loss')

plt.plot(val\_loss, label='val loss')

plt.title('Loss Graph')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

The provided code uses Matplotlib to plot two graphs side by side: one showing the training and validation accuracy over epochs, and the other showing the training and validation loss over epochs.

Live detection using VS CODE

import tensorflow as tf

# Define the emotion dictionary

emotion\_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful",

                3: "Happy", 4: "Neutral", 5: "Sad", 6: "Surprised"}

# Load the TensorFlow model

model\_path = r'emotion\_model.h5'

model = tf.keras.models.load\_model(model\_path)

This code segment loads a TensorFlow model saved in the file 'emotion\_model.h5' and defines an emotion dictionary. The dictionary maps integer labels to corresponding emotion categories.

import cv2

import numpy as np

cap = cv2.VideoCapture(0)

while True:

    # Read a frame from the video capture

    ret, frame = cap.read()

    # Resize the frame

    frame = cv2.resize(frame, (1280, 720))

    # Check if the frame was successfully read

    if not ret:

        print("Error reading frame")

        break

    # Convert the frame to grayscale

    gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

    # Detect faces in the frame

    face\_detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

    num\_faces = face\_detector.detectMultiScale(gray\_frame, scaleFactor=1.3, minNeighbors=5)

    # Process each detected face

    for (x, y, w, h) in num\_faces:

        # Draw a rectangle around the face

        cv2.rectangle(frame, (x, y-70), (x+w, y+h+18), (255,255, 102), 4)

        # Extract the region of interest (ROI) for the face

        roi\_gray\_frame = gray\_frame[y:y + h, x:x + w]

        cropped\_img = np.expand\_dims(np.expand\_dims(cv2.resize(roi\_gray\_frame, (48, 48)), -1), 0)

        # Predict the emotion for the ROI

        emotion\_prediction = model.predict(cropped\_img)

        maxindex = int(np.argmax(emotion\_prediction))

        # Display the predicted emotion as text

        cv2.putText(frame, emotion\_dict[maxindex], (x+5, y-20), cv2.FONT\_HERSHEY\_COMPLEX, 2, (254, 84, 102), 2, cv2.LINE\_AA)

    # Display the frame with bounding boxes and predicted emotions

    cv2.imshow('Emotion Detection', frame)

    # Check for key press to quit

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

        break

# Release the video capture object and close all windows

cap.release()

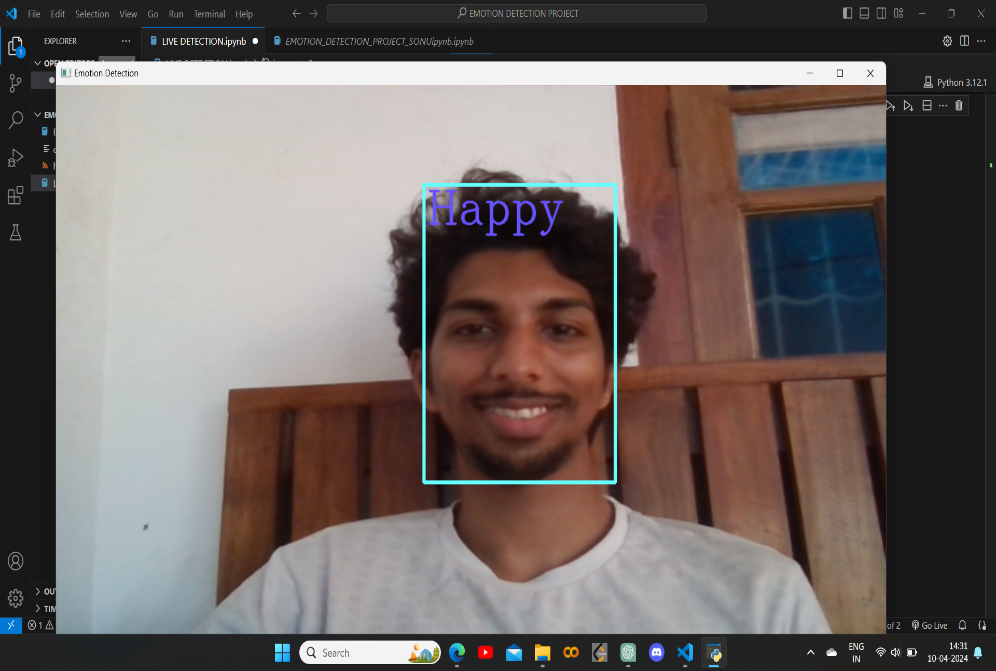
cv2.destroyAllWindows()

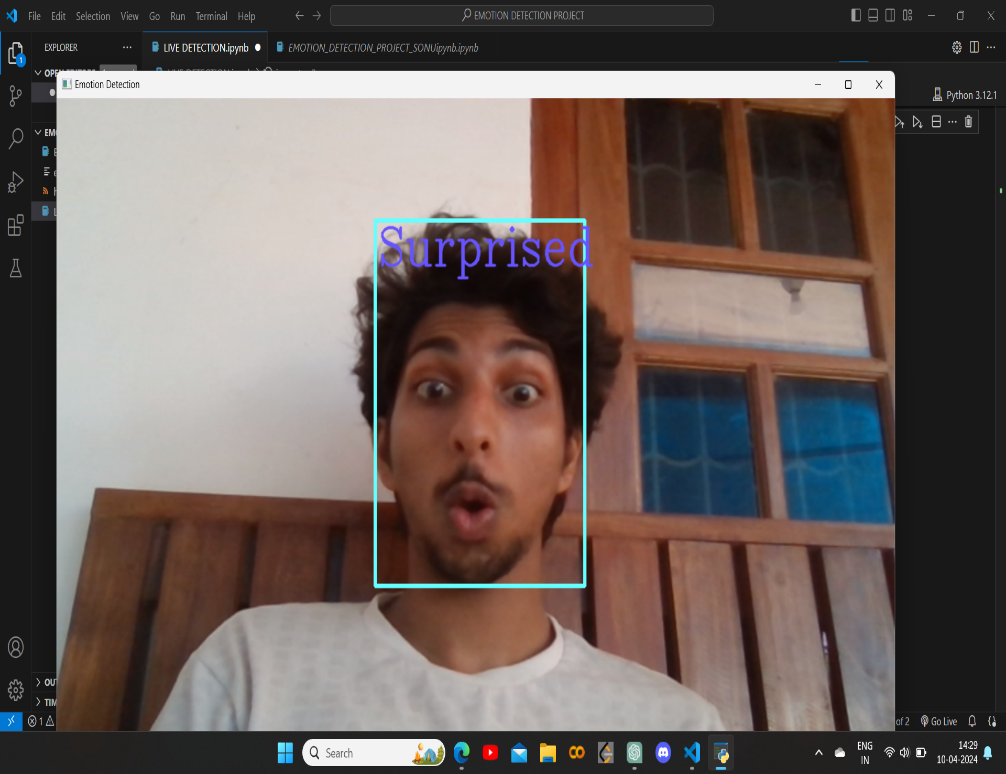
## This code segment captures video from a webcam, detects faces in the frames using Haar cascade classifiers, and predicts emotions for the detected faces using a pre-trained TensorFlow model. The predicted emotions are displayed as text on the video frames along with bounding boxes around the detected faces. The program continues running until the user presses the 'q' key to quit.

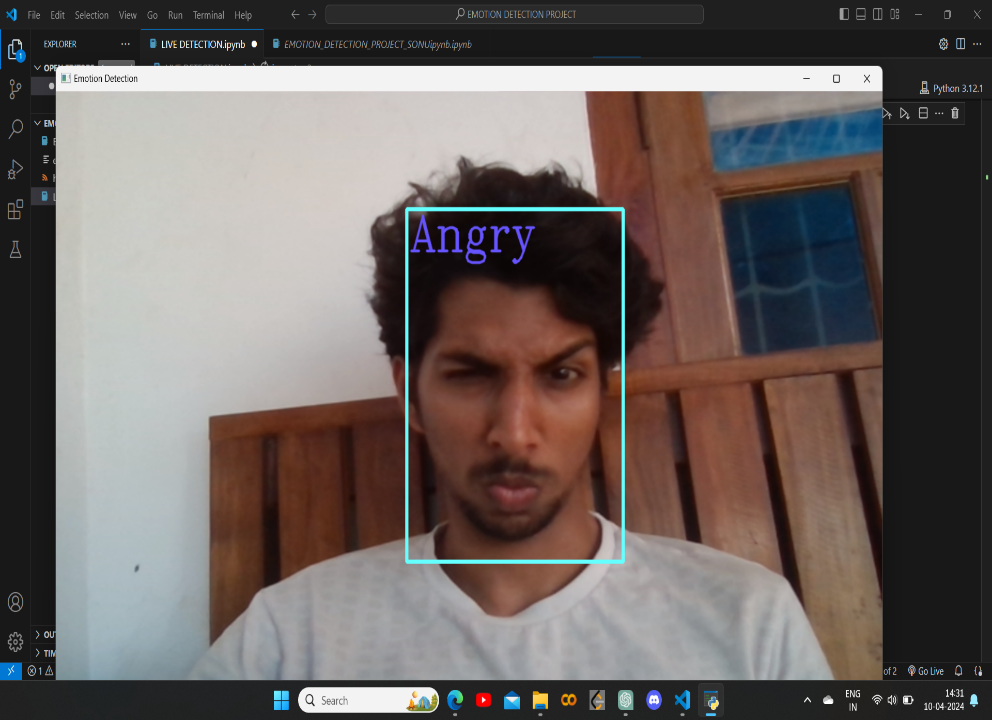
## It uses OpenCV for video capture and processing, and TensorFlow for loading and using the pre-trained emotion detection model.

## Additionally, it defines an emotion dictionary mapping integer labels to emotion categories.

Real time predictions .







* Potential Extensions:

Real-time Emotion Detection: Implement the model to perform live emotion detection from webcam feeds or video streams.

Emotion Tracking: Extend the project to track changes in emotions over time in video sequences.

Multi-face Emotion Detection: Modify the model to detect and classify emotions from multiple faces in an image or video simultaneously.

Emotion Detection in Context: Incorporate contextual information or multimodal data (e.g., audio, text) for more robust emotion detection in real-world scenarios.

* Technologies Used:

Python

TensorFlow or PyTorch (for building and training CNN models)

OpenCV (for image processing and webcam access)

Flask or Django (for web application deployment, if applicable)

TensorFlow.js or TensorFlow Lite (for deploying on web or mobile platforms, if applicable)

**Submitted by SONU PP ON 10-04-2024**