EMOTION DETECTION USING CNN

Emotions are a fundamental aspect of human experience, influencing our thoughts, behaviors, and interactions with the world around us. The ability to understand and interpret emotions is crucial in various domains, from psychology and healthcare to marketing and artificial intelligence.

Emotional detection, also known as affect recognition or emotion recognition, is the process of identifying and categorizing human emotions based on various cues such as facial expressions, voice intonation, body language, and textual content. It involves leveraging advanced technologies such as machine learning, computer vision, and natural language processing to analyze and interpret these cues.

The significance of emotional detection extends across numerous applications. In healthcare, it can aid in diagnosing mental health conditions, monitoring patient well-being, and providing personalized treatment interventions. In education, it can facilitate adaptive learning systems that respond to students' emotional states, optimizing the learning process. In customer service, it can enhance user experiences by enabling empathetic interactions and resolving issues more effectively.

Moreover, emotional detection plays a pivotal role in human-computer interaction, enabling computers to perceive and respond to users' emotional states. This capability is increasingly important in the development of emotionally intelligent systems, virtual assistants, and social robots that can engage with users in more natural and empathetic ways.

TYPES OF EMOTION DETECTION

Emotion detection can be categorized into several types based on the modalities or channels through which emotions are expressed and detected. Here are some common types of emotion detection:

Facial Expression Analysis

This type of emotion detection focuses on analyzing facial expressions to infer emotional states. It involves capturing and analyzing facial features such as the movement of facial muscles, changes in expression, and micro-expressions that occur within milliseconds. Facial expression analysis is widely used in fields such as psychology, human-computer interaction, and market research.

Speech and Voice Analysis

Emotions can also be conveyed through speech and vocal cues such as tone, pitch, intensity, and rhythm. Speech and voice analysis techniques involve extracting features from audio recordings to detect emotional states such as happiness, sadness, anger, or excitement. This type of emotion detection is commonly used in call centers, virtual assistants, and sentiment analysis of audio content.

Textual Analysis

Written text, including emails, social media posts, chat messages, and product reviews, contains valuable cues about the emotional state of the author. Textual analysis techniques, often referred to as sentiment analysis or affective computing, involve processing and analyzing text to identify sentiments, emotions, and attitudes expressed by the author. Textual analysis has applications in social media monitoring, customer feedback analysis, and content moderation.

Physiological Signals Analysis

Emotions are associated with physiological changes in the body, such as heart rate variability, skin conductance, and brain activity. Physiological signals analysis

techniques involve measuring and interpreting these physiological signals to infer emotional states. This type of emotion detection is used in fields such as healthcare, neurology, and human-computer interaction to monitor emotional arousal, stress levels, and mental well-being.

Multimodal Emotion Detection

Multimodal emotion detection combines information from multiple modalities, such as facial expressions, speech, text, and physiological signals, to improve the accuracy and robustness of emotion detection systems. By integrating data from different channels, multimodal approaches can capture a more comprehensive understanding of human emotions and behaviors. Multimodal emotion detection is particularly useful in real-world applications where emotions are expressed through multiple channels simultaneously.

METHODOLOGY

Emotion detection can be approached using various methodologies, each with its own advantages and suitability for different applications. Here are several methodologies commonly used for emotion detection:

Machine Learning Algorithms

Machine learning algorithms, including traditional classifiers such as Support Vector Machines (SVM), Random Forests, and k-Nearest Neighbors (k-NN), can be trained on labeled datasets of emotional expressions to classify new instances into predefined emotion categories. These algorithms are suitable for tasks where feature engineering plays a crucial role in extracting relevant information from input data.

Deep Learning Models

Deep learning models, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown remarkable success in various emotion detection tasks, especially in the domains of computer vision and natural language processing. CNNs excel in analyzing visual data such as images or videos, while RNNs are well-suited for sequential data such as speech or text. Deep learning models can automatically learn hierarchical representations of input data, eliminating the need for manual feature extraction.

Facial Expression Analysis

Facial expression analysis focuses on extracting features from facial images or videos to infer emotional states. This methodology involves techniques such as facial landmark detection, feature extraction (e.g., from facial action units), and machine learning or deep learning-based classification. Facial expression analysis is widely used in applications ranging from human-computer interaction to mental health monitoring.

Speech and Voice Analysis

Emotions can be conveyed through speech and vocal cues, making speech and voice analysis a valuable methodology for emotion detection. This approach involves extracting acoustic features from audio recordings, such as pitch, intensity, and spectral characteristics, and applying machine learning or deep learning techniques to classify emotional states. Speech and voice analysis are commonly used in call centers, virtual assistants, and sentiment analysis tasks.

Textual Analysis

Textual analysis, also known as sentiment analysis or affective computing, focuses on analyzing written text to infer emotional states or sentiment. This methodology involves natural language processing techniques such as text preprocessing, feature extraction (e.g., word embeddings), and classification algorithms (e.g., Support Vector Machines, Recurrent Neural Networks). Textual analysis is widely used in

applications such as social media monitoring, customer feedback analysis, and content moderation.

Multimodal Fusion

Multimodal fusion combines information from multiple modalities, such as facial expressions, speech, text, and physiological signals, to improve the accuracy and robustness of emotion detection systems. This methodology leverages techniques such as feature-level fusion, decision-level fusion, or late fusion to integrate information from different modalities and make collective predictions about emotional states.

These are just a few examples of methodologies used in emotion detection. Depending on the specific requirements of a task or application, researchers and practitioners may employ a combination of these methodologies to achieve the desired outcomes.

INTRODUCTION TO PROJECT

We have outlined the importance,,various types and mathodologies of emotional detection.Now let's us focuses on facial recognition for emotional detection. Our project aims to harness the power of facial expressions as a key indicator of human emotions and develop an innovative emotion recognition system.

OBJECTIVE

In this project, our primary objective is to develop a sophisticated emotion recognition system that can accurately analyze facial expressions and infer underlying emotional states such as sad,anger,happy,surprise,fear and neutral. By integrating cutting-edge technologies and methodologies, including deep learning

algorithms and computer vision techniques, we aim to create a robust system capable of recognizing a wide spectrum of emotions with high precision.

TECHNICAL TERMS

A Convolutional Neural Network (CNN) is a type of deep learning algorithm inspired by the structure and function of the human visual cortex. It's designed to automatically and adaptively learn hierarchical patterns or features from input data, particularly in the domain of computer vision.

The key components of a CNN include:

Convolutional Layers

These layers apply learnable filters (also called kernels) to the input data, typically images. The filters convolve across the input data, extracting spatial patterns or features. Each filter detects specific patterns, such as edges, textures, or shapes, and produces a feature map as output.

Pooling Layers

Pooling layers reduce the spatial dimensions of the feature maps produced by the convolutional layers. Common pooling operations include max pooling, which selects the maximum value within each pooling region, and average pooling, which computes the average value. Pooling helps to make the representations learned by the network more invariant to small spatial translations and distortions.

Activation Functions

Activation functions introduce non-linearities into the network, allowing it to capture complex relationships in the data. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh. ReLU is widely used in CNNs due to its simplicity and effectiveness in mitigating the vanishing gradient problem.

Fully Connected Layers

Fully connected layers, also known as dense layers, connect every neuron in one layer to every neuron in the next layer. These layers typically appear towards the end of the network and are responsible for combining the high-level features learned by the convolutional layers into class scores or predictions.

Loss Function

The loss function quantifies the difference between the predicted output of the network and the true labels in the training data. Common loss functions for classification tasks include categorical cross-entropy and softmax cross-entropy.

During training, a CNN learns to automatically extract relevant features from the input data through the process of backpropagation and gradient descent optimization. By iteratively adjusting the parameters of the network based on the difference between predicted and actual outputs, the CNN gradually improves its ability to accurately classify or analyze images.

CNNs have demonstrated remarkable success in various computer vision tasks, including image classification, object detection, facial recognition, and more. Their ability to learn hierarchical representations of visual data makes them well-suited for a wide range of real-world applications.

TECHNICAL CODE

REQUIRED MODULES

These are all the modules required to build our convolutional neural network

Tensorflow	Keras

Numpy	Pandas
Matplotlib	OpenCv
Scipy	tqdm

IMPORTING LIBRARIES

```
import matplotlib.pyplot as plt
import numpy as np
import scipy
import pandas as pd
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
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers.schedules import ExponentialDecay
import cv2
```

This includes the necessary libraries and modules for building and training a neural network. It uses Keras for building the model, pandas for handling data in a tabular form, and other modules for various functionalities.

IMPORTING DATASET

```
import zipfile

zip_ref = zipfile.ZipFile('/content/drive/MyDrive/Colab Notebooks/emot.zip', 'r')

zip_ref.extractall('/content')

zip_ref.close()
```

import zipfile: This imports the zipfile module, which provides tools for creating, reading, writing, and extracting ZIP files in Python.

zip_ref=zipfile.ZipFile('/content/drive/MyDrive/Colab Notebooks/emot.zip', 'r'): This line creates a ZipFile object named zip_ref, specifying the path to the ZIP file (/content/drive/MyDrive/Colab Notebooks/emot.zip) and opening it in read mode ('r').

zip_ref.extractall('/content'): This line extracts all the contents of the ZIP file to the /content directory. The extractall() method extracts all files and folders from the archive to the specified directory.

zip_ref.close(): This line closes the ZipFile object, releasing any resources associated with it.

INITILIZATION

The rescale parameter is used to normalize the pixel values of the input images by dividing pixel value by 255.

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

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

PREPROCESSING TRAIN AND TEST IMAGES

```
train_generator = train_data_gen.flow_from_directory(
```

```
    '/content/train',
    target_size=(48, 48),
    batch_size=64,
    color_mode="grayscale",
    class_mode='categorical')
```

The flow_from_directory method from the ImageDataGenerator class in Keras to generate batches of image data from a directory. Here's a breakdown of the parameters you've used:

- /content/train: This specifies the directory containing your training images. The method will scan this directory for images and their corresponding labels.
- target_size=(48, 48): This parameter specifies the dimensions to which all images will be resized. In this case, images will be resized to a square shape with dimensions of 48x48 pixels.
- batch_size=64: This parameter determines the size of the batches of images that will be generated during training. Each batch will contain 64 images.
- color_mode="grayscale": This parameter specifies the color mode of the images. "grayscale" indicates that the images will be converted to grayscale, meaning they will have a single channel representing intensity rather than three channels (RGB).
- class_mode='categorical': This parameter specifies the type of label arrays that are returned by the generator. "categorical" indicates that the labels are represented as one-hot encoded categorical arrays.

Output: Found 28709 images belonging to 7 classes

```
validation_generator = validation_data_gen.flow_from_directory(
    '/content/test',
    target_size=(48, 48),
    batch_size=64,
    color_mode="grayscale",
    class_mode='categorical')
```

The flow_from_directory method are similar to how you set up the training data generator.

Output: Found 7178 images belonging to 7 classes

CREATING CNN

```
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'])
```

Here's a breakdown of your model architecture and compilation:

1.Model Architecture:

- You're using a Sequential model, which is a linear stack of layers.
- The first layer is a Conv2D layer with 32 filters, each with a 3x3 kernel size, and ReLU activation function. The input shape is set to (48, 48, 1), indicating grayscale images with dimensions 48x48 pixels.
- This is followed by another Conv2D layer with 64 filters and ReLU activation.

- Next, a MaxPooling2D layer with a 2x2 pool size is applied to reduce spatial dimensions.
- Dropout regularization with a rate of 0.25 is added to mitigate overfitting.
- Additional Conv2D layers with 128 filters and ReLU activation, followed by MaxPooling2D layers, are added.
- Another Dropout layer with a rate of 0.25 is included.
- The Flatten layer converts the 2D feature maps into a 1D vector.
- Dense layers with 1024 units and ReLU activation, followed by Dropout with a rate of 0.5, are added.
- Finally, a Dense layer with 7 units (for 7 emotion classes) and softmax activation function is added for classification.

2. Model Summary:
    - The summary of the model architecture is printed, showing the structure of each layer and the number of parameters.
3. Optimizer and Learning Rate Schedule:
    - Adam optimizer is used with an initial learning rate of 0.0001.
    - ExponentialDecay learning rate schedule is applied, which exponentially decays the learning rate over time.
    - This learning rate schedule helps improve convergence during training.
4. Compilation:
    - The model is compiled using categorical cross-entropy loss, suitable for multi-class classification tasks.
    - Adam optimizer with the specified learning rate schedule is used.
    - Metrics for evaluation are set to accuracy.

OUTPUT

Model: "sequential"

_____

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 46, 46, 32)	320
conv2d_1 (Conv2D)	(None, 44, 44, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
conv2d_2 (Conv2D)	(None, 20, 20, 128)	73856
max_pooling2d_1 (MaxPoolin	(None, 10, 10, 128)	0

g2D)

conv2d_3 (Conv2D) (None, 8, 8, 128) 147584

max_pooling2d_2 (MaxPoolin (None, 4, 4, 128) 0

g2D)

dropout_1 (Dropout) (None, 4, 4, 128) 0

flatten (Flatten) (None, 2048) 0

dense (Dense) (None, 1024) 2098176

dropout_2 (Dropout) (None, 1024) 0

dense_1 (Dense) (None, 7) 7175

==========

======

Total params: 2345607 (8.95 MB)

Trainable params: 2345607 (8.95 MB)

Non-trainable params: 0 (0.00 Byte)

TRAINING

```
emotion_model_info=emotion_model.fit(train_generator,steps_per_epoch=28709 // 64,
        epochs=30,validation_data=validation_generator,validation_steps=7178     // 64)
```

1. train_generator:
    - This is the generator object that yields batches of training data. It's typically created using the flow_from_directory method of an ImageDataGenerator object and provides images and labels for training.
2. steps_per_epoch:
    - This parameter specifies the number of steps (batches) to yield from the train_generator **g**enerator for each epoch. It's typically calculated as the total number of training samples divided by the batch size.
3. epochs:
    - This parameter specifies the number of epochs (iterations over the entire dataset) for which the model will be trained.
4. validation_data:

- This parameter specifies the generator object (validation_generator) that yields batches of validation data. Similar to train_generator**,** it's typically created using the flow_from_directory method of an ImageDataGenerator object and provides images and labels for validation.

5. validation_steps:
    - This parameter specifies the number of steps (batches) to yield from the validation_generator generator for each epoch during validation. It's typically calculated as the total number of validation samples divided by the batch size.

OUTPUT

Epoch 1/30 448/448 [==============================] - 16s 35ms/step - loss: 0.7074 - accuracy: 0.7423 - val_loss: 1.0933 - val_accuracy: 0.6071

Epoch 2/30 448/448 [==============================] - 16s 36ms/step - loss: 0.6966 - accuracy: 0.7467 - val_loss: 1.0867 - val_accuracy: 0.6078

Epoch 3/30 448/448 [==============================] - 16s 35ms/step - loss: 0.6693 - accuracy: 0.7585 - val_loss: 1.0903 - val_accuracy: 0.6124

Epoch 4/30 448/448 [==============================] - 14s 31ms/step - loss: 0.6467 - accuracy: 0.7662 - val_loss: 1.0933 - val_accuracy: 0.6154

Epoch 5/30 448/448 [==============================] - 15s 33ms/step - loss: 0.6313 - accuracy: 0.7714 - val_loss: 1.0996 - val_accuracy: 0.6097

Epoch 6/30 448/448 [==============================] - 14s 31ms/step - loss: 0.6050 - accuracy: 0.7792 - val_loss: 1.1023 - val_accuracy: 0.6115

Epoch 7/30 448/448 [==============================] - 14s 31ms/step - loss: 0.5905 - accuracy: 0.7877 - val_loss: 1.0914 - val_accuracy: 0.6145

Epoch 8/30 448/448 [==============================] - 14s 32ms/step - loss: 0.5649 - accuracy: 0.7951 - val_loss: 1.1223 - val_accuracy: 0.6190

Epoch 9/30 448/448 [==============================] - 17s 37ms/step - loss: 0.5494 - accuracy: 0.8025 - val_loss: 1.1168 - val_accuracy: 0.6177

Epoch 10/30 448/448 [==============================] - 14s 31ms/step - loss: 0.5329 - accuracy: 0.8072 - val_loss: 1.1362 - val_accuracy: 0.6233

Epoch 11/30 448/448 [==============================] - 16s 36ms/step - loss: 0.5190 - accuracy: 0.8112 - val_loss: 1.1250 - val_accuracy: 0.6236

Epoch 12/30 448/448 [==============================] - 16s 36ms/step - loss: 0.5050 - accuracy: 0.8152 - val_loss: 1.1424 - val_accuracy: 0.6191

Epoch 13/30 448/448 [==============================] - 14s 32ms/step - loss: 0.4895 - accuracy: 0.8226 - val_loss: 1.1481 - val_accuracy: 0.6130

Epoch 14/30 448/448 [==============================] - 14s 31ms/step - loss: 0.4688 - accuracy: 0.8319 - val_loss: 1.1751 - val_accuracy: 0.6177

Epoch 15/30 448/448 [==============================] - 14s 32ms/step - loss: 0.4544 - accuracy: 0.8361 - val_loss: 1.1575 - val_accuracy: 0.6233

Epoch 16/30 448/448 [==============================] - 14s 32ms/step - loss: 0.4438 - accuracy: 0.8384 - val_loss: 1.1862 - val_accuracy: 0.6191

Epoch 17/30 448/448 [==============================] - 14s 31ms/step - loss: 0.4257 - accuracy: 0.8471 - val_loss: 1.2062 - val_accuracy: 0.6247

Epoch 18/30 448/448 [==============================] - 14s 31ms/step - loss: 0.4149 - accuracy: 0.8506 - val_loss: 1.1950 - val_accuracy: 0.6258

Epoch 19/30 448/448 [==============================] - 14s 32ms/step - loss: 0.3955 - accuracy: 0.8560 - val_loss: 1.2086 - val_accuracy: 0.6196

Epoch 20/30 448/448 [==============================] - 15s 34ms/step - loss: 0.3886 - accuracy: 0.8596 - val_loss: 1.2045 - val_accuracy: 0.6261

Epoch 21/30 448/448 [==============================] - 16s 36ms/step - loss: 0.3805 - accuracy: 0.8592 - val_loss: 1.2117 - val_accuracy: 0.6249

Epoch 22/30 448/448 [==============================] - 14s 32ms/step - loss: 0.3644 - accuracy: 0.8698 - val_loss: 1.2118 - val_accuracy: 0.6217

Epoch 23/30 448/448 [==============================] - 15s 33ms/step - loss: 0.3594 - accuracy: 0.8718 - val_loss: 1.2077 - val_accuracy: 0.6274

Epoch 24/30 448/448 [==============================] - 17s 38ms/step - loss: 0.3491 - accuracy: 0.8757 - val_loss: 1.2452 - val_accuracy: 0.6247

Epoch 25/30 448/448 [==============================] - 14s 31ms/step - loss: 0.3307 - accuracy: 0.8812 - val_loss: 1.2696 - val_accuracy: 0.6222

Epoch 26/30 448/448 [==============================] - 14s 32ms/step - loss: 0.3255 - accuracy: 0.8825 - val_loss: 1.2461 - val_accuracy: 0.6271

Epoch 27/30 448/448 [==============================] - 14s 32ms/step - loss: 0.3237 - accuracy: 0.8848 - val_loss: 1.2522 - val_accuracy: 0.6225

Epoch 28/30 448/448 [==============================] - 14s 32ms/step - loss: 0.3070 - accuracy: 0.8908 - val_loss: 1.2921 - val_accuracy: 0.6205

Epoch 29/30 448/448 [==============================] - 14s 32ms/step - loss: 0.3029 - accuracy: 0.8919 - val_loss: 1.2887 - val_accuracy: 0.6286

Epoch 30/30 448/448 [==============================] - 15s 35ms/step - loss: 0.2948 - accuracy: 0.8942 - val_loss: 1.2997 - val_accuracy: 0.6228

Each line represents one epoch of training and validation. For example:

- In Epoch 1/30, the training loss is 0.7074, training accuracy is 0.7423, validation loss is 1.0933, and validation accuracy is 0.6071.
- In Epoch 2/30, the training loss is 0.6966, training accuracy is 0.7467, validation loss is 1.0867, and validation accuracy is 0.6078.
- And so on, for all 30 epochs.

SAVING THE MODEL

By using the save method with the filename 'emotion_model.h5', you've saved the entire architecture and weights of your trained model to a single HDF5 file. This file can be later loaded and used for inference or further training if needed.

The line print("Model saved successfully") is a helpful confirmation message indicating that the model was saved without any errors.

```
emotion_model.save('emotion_model.h5')

print("Model saved successfully")
```

ACCURACY AND LOSS EVALUATION

It seems like you're using the evaluate method to assess the performance of your emotion detection model on the validation dataset provided by validation_generator. This method computes the loss value and metrics (such as accuracy) for the model on the validation data.

```
emotion_model.evaluate(validation_generator)
```

OUTPUT:

113/113 [==============================] - 3s 25ms/step - loss: 1.0847 - accuracy: 0.6055

[1.0846856832504272, 0.6054611206054688]

VISUALISING ACCURACY AND LOSS

```
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']
```

Here's what each line does:

1. accuracy = emotion_model_info.history['accuracy']: Extracts the training accuracy values from the history object and stores them in the variable accuracy.

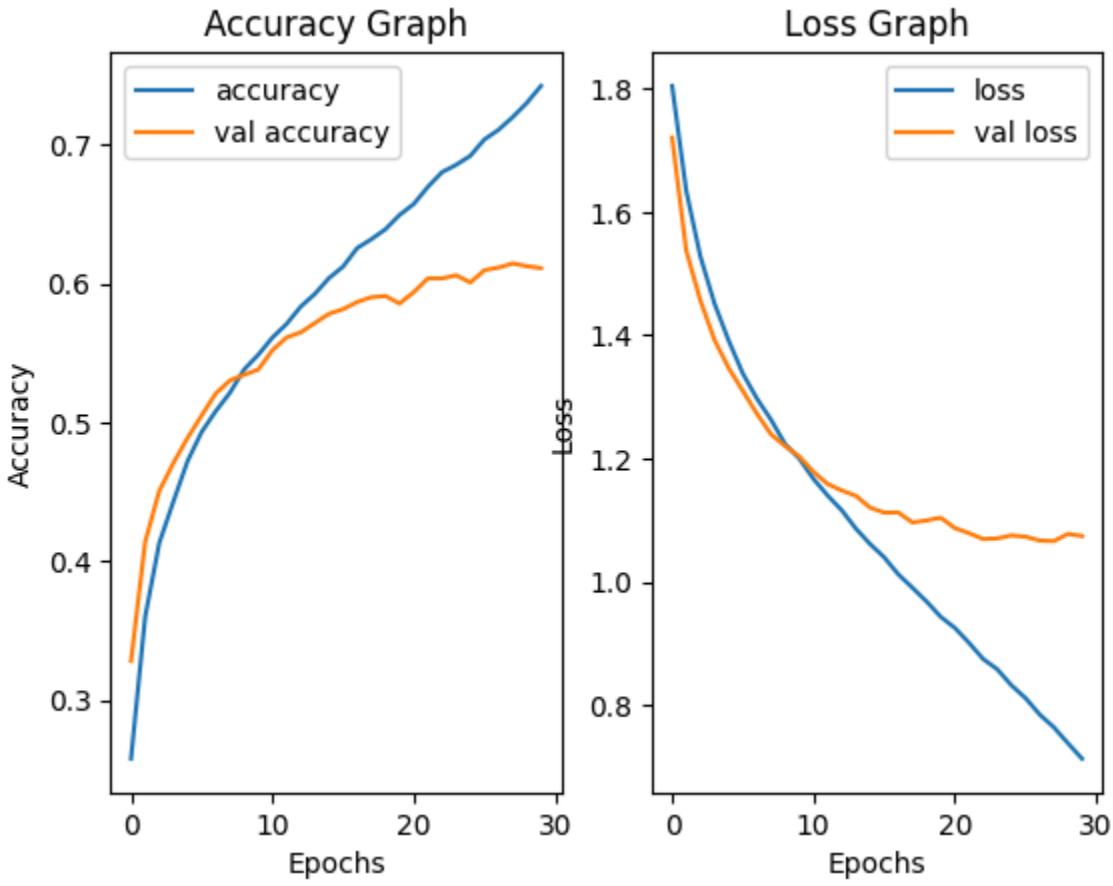
2. val_accuracy = emotion_model_info.history['val_accuracy']: Extracts the validation accuracy values from the history object and stores them in the variable val_accuracy.
3. loss = emotion_model_info.history['loss']: Extracts the training loss values from the history object and stores them in the variable loss.
4. val_loss = emotion_model_info.history['val_loss']: Extracts the validation loss values from the history object and stores them in the variable val_loss.

These variables now contain the training and validation metrics for accuracy and loss across epochs, which you can use for visualizations.

GRAPH

```
import matplotlib.pyplot as plt
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()
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()
```



PREDICTION

The code creates key-value pair in the dictionary represents a unique emotion label (0 to 6) mapped to its corresponding emotion category.

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

IMPLEMENTATION USING OPENCV

Here we are implementing a real-time emotion detection system using OpenCV and a pre-trained model.

```
cap = cv2.VideoCapture(0)

while True:

    ret, frame = cap.read()

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

     if not ret:

        print(ret)

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

    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            num_faces = face_detector.detectMultiScale(gray_frame, 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 = 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)

        emotion_prediction = emotion_model.predict(cropped_img)

        maxindex = int(np.argmax(emotion_prediction))

        cv2.putText(frame, emotion_dict[maxindex], (x+5, y-20),
```

```
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)

    cv2.imshow('Emotion Detection', frame)

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

       Break

cap.release()

cv2.destroyAllWindows()
```