มหาวิทยาลัยอัสสัมชัญ
ASSUMPTION UNIVERSITY
of THAILAND

VINCENT MARY
SCHOOL of SCIENCE and
TECHNOLOGY

ASSUMPTION UNIVERSITY

Vincent Mary School of Science and Technology

Bachelor of Science Program in Information Technology

CSX4201 AI Concept

Emotion Classifier

Project Final Report

By

Siwach Toprasert	6316801
Saw Zwe Wai Yan	6318013

Advisor: Asst. Prof. Dr. Thitipong Tanprasert

ABSTRACT

Emotions play a fundamental role in human communication and perception, influencing our behavior and decision-making processes. The ability to accurately detect and classify emotions from visual cues is a crucial task with applications ranging from human-computer interaction to healthcare. In recent years, advances in deep learning and computer vision techniques have paved the way for the development of emotion classifiers. This paper presents an Emotion Classifier that leverages deep neural networks to automatically recognize and classify human emotions in facial expressions.

Keywords: Emotion Classifier, deep neural networks, facial expressions, data preprocessing, feature extraction, mental health assessment, artificial intelligence concepts

Table of Contents

Table of Figures

1 INTRODUCTION

In an increasingly digitized world, the ability to understand and interpret human emotions plays a pivotal role in various domains, from improving user experiences in technology to enhancing the effectiveness of marketing strategies and even assisting in mental health diagnostics. The power of emotion recognition, primarily through machine learning algorithms, such as deep learning, has opened up exciting possibilities for automated systems to comprehend and respond to human emotions. This project embarks on the journey of developing an Emotion Classifier that leverages deep neural networks to accurately detect and classify emotions based on facial expressions. Emotions, a complex interplay of facial features and expressions, are fundamental to human communication and interaction. Recognizing these emotional cues with precision and efficiency is not only a technological challenge but also a critical step towards creating empathetic and responsive systems. In this project, we explore the architecture, methodologies, and applications of our Emotion Classifier, highlighting its potential to revolutionize fields ranging from human-computer interaction to mental health assessment.

2 OBJECTIVES

The primary objective of this project is to design, develop, and evaluate an advanced Emotion Classifier to achieve the following:

I. **Accurate Emotion Recognition**: Create a robust system capable of accurately recognizing and classifying a wide range of human emotions based on facial expressions, including but not limited to happiness, sadness, anger, fear, disgust, and surprise.

II. **Generalization**: Ensure that the Emotion Classifier can generalize well across diverse demographics, including age groups, genders, and ethnicities, by training on large and varied datasets.

III. **Efficiency**: Develop an efficient and real-time capable system that can process and analyze facial expressions in a timely manner, making it suitable for interactive applications such as human-computer interaction and user experience customization.

IV. **Applications**: Explore the practical applications of the Emotion Classifier in fields such as mental health assessment through automated emotion monitoring.

V. **Performance Evaluation**: Conduct comprehensive performance evaluations and benchmarking against existing emotion recognition methods and datasets to assess the accuracy, precision, and reliability of the Emotion Classifier.

VI. **Documentation and Knowledge Sharing**: Document the development process, methodologies, and results comprehensively to facilitate knowledge sharing and future research in the field of affective computing.

By achieving these objectives, this project aims to contribute to the advancement of emotion recognition technology and its broader adoption in applications that benefit from understanding and responding to human emotions.

2.1 SCOPE

Many organizations have been using a paper-based system for quite a while. This can be a significant pain point for their stakeholders. A paper-based system is often slow, inefficient, and prone to errors. It requires manual data entry and handling, which can be time-consuming and tedious for staff members. Paper-based systems can also be difficult to manage and track, as documents can be easily lost or misplaced.

Moreover, paper-based systems are not easily scalable and can become unwieldy as the organization grows. This can lead to additional costs and resource allocation for storage and management of physical documents. Additionally, paper-based systems can be vulnerable to data breaches or unauthorized access, as it can be difficult to control who has access to sensitive information. Lastly, it can lead to delays, errors, and inefficiencies, which can have a negative impact on both the organization and its stakeholders The scope of the Emotion Classifier project encompasses the following key areas:

I. **Emotion Recognition**: The project will focus on developing an Emotion Classifier to accurately recognize and classify a range of human emotions based on facial expressions. Emotions of interest include, but are not limited to, happiness, sadness, anger, fear, disgust, and surprise.

II. **Data Collection and Preparation**: The project will involve the acquisition and preparation of diverse and representative datasets containing annotated facial expressions to train and evaluate the Emotion Classifier. Efforts will be made to ensure data quality and diversity to enhance the model's robustness.

III. **Algorithm Development**: The core of the project involves designing and implementing advanced algorithms to extract relevant features from facial images and classify emotions effectively. Techniques such as convolutional neural networks (CNNs), data augmentation, and transfer learning may be employed.

IV. **Generalization and Evaluation**: The Emotion Classifier's generalization across demographics, including age groups, genders, and ethnicities, will be a significant focus. Extensive performance evaluations and benchmarking against existing methods will be conducted to assess accuracy, precision, and reliability.

V. **Documentation**: Comprehensive documentation of the development process, methodologies, and results will be maintained to facilitate knowledge sharing and future research in the field of affective computing.

2.2 LIMITATIONS

While the Emotion Classifier project aims to provide accurate emotion recognition, it is essential to acknowledge its limitations:

I. **Limited Accuracy in Complex Contexts**: Emotion recognition solely based on facial expressions may struggle to capture emotions in complex real-world contexts where multiple factors, such as tone of voice and body language, also contribute to emotional understanding.

II. **Overcoming Occlusions**: The Emotion Classifier may face challenges when facial expressions are partially obscured, such as when people wear masks, glasses, or have heavy facial hair, affecting its accuracy.

III. **Data Bias**: The performance of the classifier heavily relies on the quality and diversity of the training data. If the training data is biased or unrepresentative, the model may exhibit biases and limitations in recognizing emotions.

3 LITERATURE REVIEWS

Research in facial emotion recognition has witnessed substantial growth over the years, driven by the increasing importance of understanding human emotions in various domains. Notable studies have applied a range of techniques, from traditional machine learning to deep learning, to analyze facial expressions and classify emotions accurately. Prior works have utilized datasets like the CK+ database and FER2013 to train and evaluate emotion recognition models. These datasets have contributed significantly to the development of robust systems.

The "Depression Dataset on Facial Expression Images" (available at Kaggle) stands out as a promising resource in the realm of emotion recognition. This dataset, curated explicitly for emotion analysis in the context of depression assessment, comprises a diverse collection of facial images. It categorizes emotions into seven distinct classes, including happiness, sadness, anger, fear, disgust, surprise, and neutral expressions. The dataset's careful curation ensures a rich and varied source of emotional facial data.

Our Emotion Classifier project aims to contribute to this growing body of research by harnessing the potential of the "Depression Dataset on Facial Expression Images." By training and evaluating our model using this dataset, we seek to advance the accuracy and effectiveness of emotion recognition, particularly in scenarios related to mental health assessment. The project's focus on image processing and deep learning techniques aligns with methods employed in recent research.

4 METHODOLOGIES

The development of an Emotion Classifier involves several key methodologies and steps to ensure accuracy and efficiency in emotion recognition. This chapter delves into the methodologies considered during the development of the program.

4.1 DATA COLLECTION AND PREPROCESSING

As mentioned in the last section, the data for this project was acquired from the Kaggle dataset titled " Depression Dataset on Facial Expression Images" retrieved from Kaggle. This dataset offers a valuable collection of labeled facial images specifically designed for emotion recognition in the context of depression assessment. It contains a diverse array of facial expressions, categorized into seven distinct emotion classes, including happiness, sadness, anger, fear, disgust, surprise, and neutral expressions. The dataset has been thoughtfully curated to provide a rich and varied source of emotional facial data, making it an ideal resource for training, and evaluating the Emotion Classifier. By leveraging this dataset, we aim to enhance the accuracy and effectiveness of our emotion recognition system while contributing to advancements in the field of mental health assessment through automated emotion analysis.

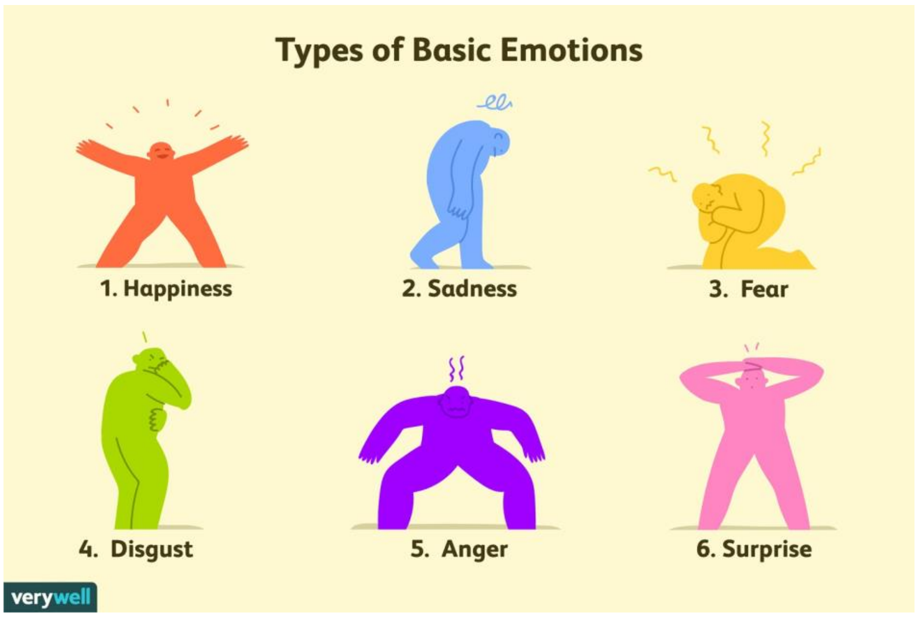
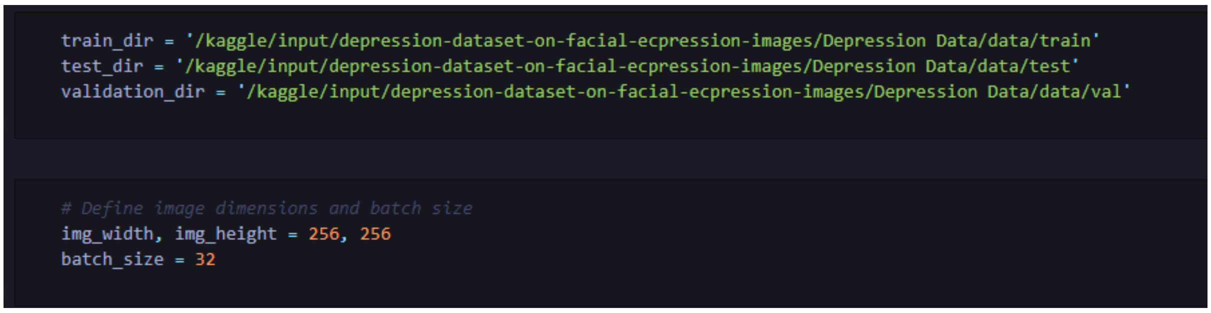


Figure 1: Types of Basic Emotions

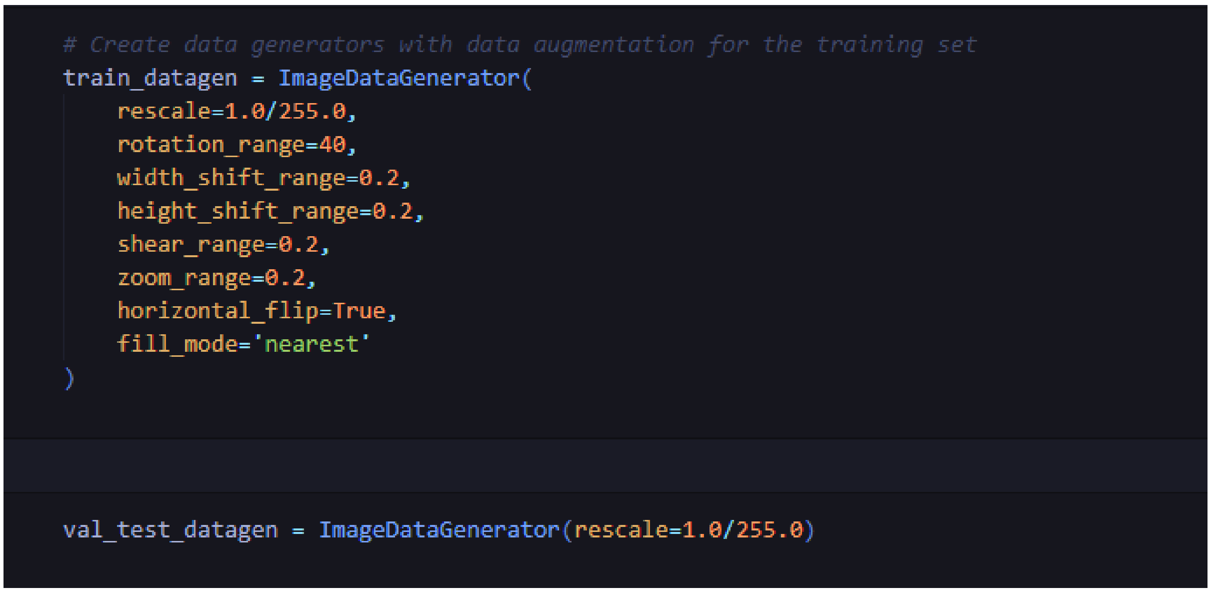
4.2 MODEL TRAINING



```python
train_dir = '/kaggle/input/depression-dataset-on-facial-ecpression-images/Depression Data/data/train'
test_dir = '/kaggle/input/depression-dataset-on-facial-ecpression-images/Depression Data/data/test'
validation_dir = '/kaggle/input/depression-dataset-on-facial-ecpression-images/Depression Data/data/val'

# Define image dimensions and batch size
img_width, img_height = 256, 256
batch_size = 32
```

Figure 2: Load Data

The data is separated into three categories; train set, test set, and validation set. In Figure 3, the directory of the images for the train set, test set, and validation set is defined respectively. The images are expected to have dimensions of 256 pixels in width and height. Additionally, the 'batch_size' is set to 32, indicating that during model training, 32 images will be processed together in each iteration or batch. This code serves as a crucial setup for loading and processing image data in subsequent machine learning tasks.



```python
# Create data generators with data augmentation for the training set
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_test_datagen = ImageDataGenerator(rescale=1.0/255.0)
```

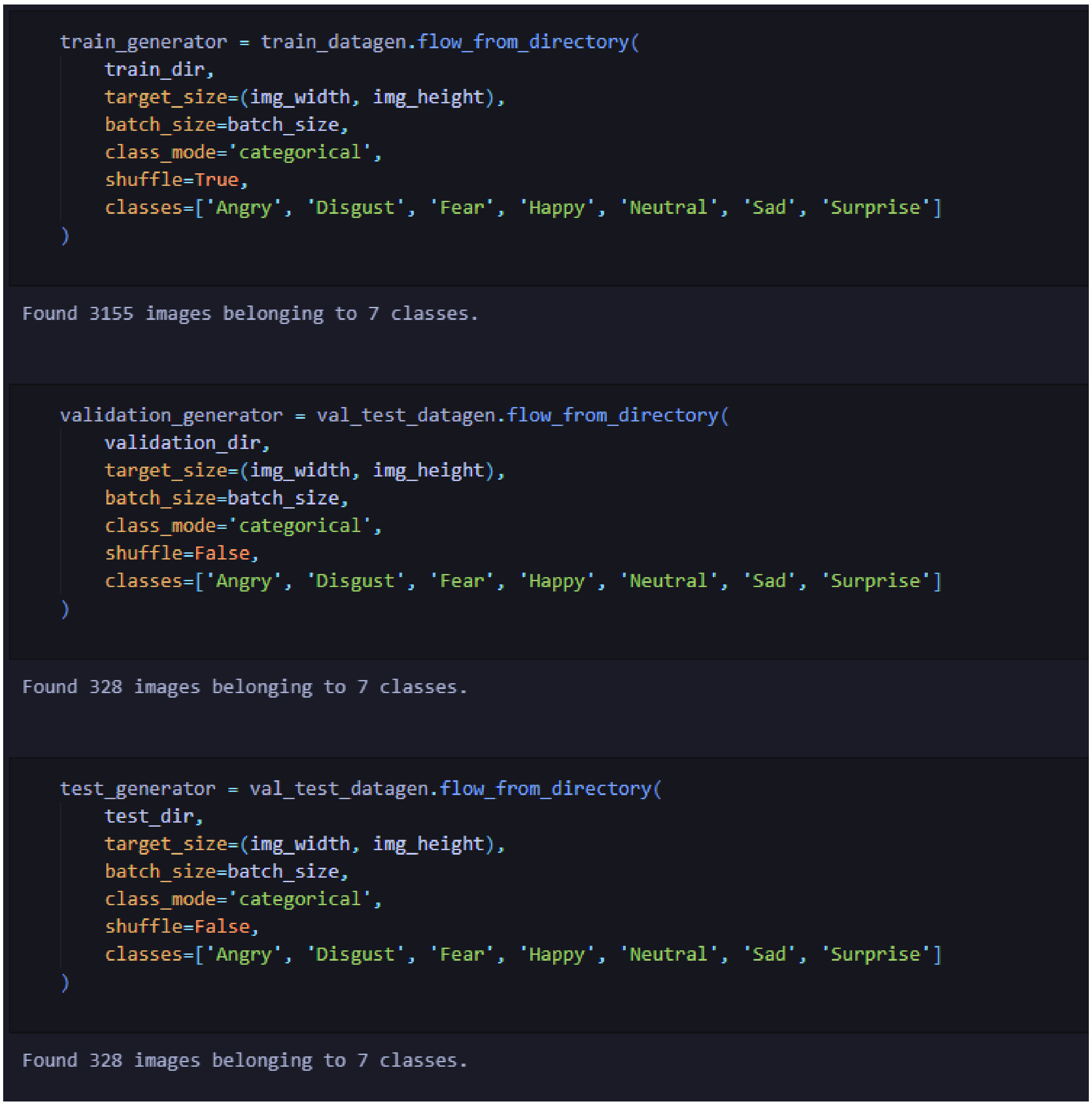
Figure 3: Data Generators with Data Augmentation

Figure 4 defines the data generator for train set, test set, and validation set. The first generator, ‘train_datagen’, is configured with various data augmentation techniques, including rotation, width and height shifting, shearing, zooming, and horizontal flipping. These techniques help diversify the training data, making the model more robust and less prone to overfitting.

Additionally, the images are rescaled by dividing each pixel value by 255 to bring them into the range [0, 1]. This is a common practice in image processing to ensure consistent scaling.

The second generator, ‘val_test_datagen’, is intended for the validation and test datasets. It only performs rescaling, similar to the first generator, to ensure that the pixel values are within the [0, 1] range.

These data generators will be used in conjunction with image directories to load and preprocess images before feeding them into a machine learning model, typically a deep neural network, for training and evaluation.



```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True,
    classes=['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']
)
```

```
Found 3155 images belonging to 7 classes.
```

```
validation_generator = val_test_datagen.flow_from_directory(
    validation_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False,
    classes=['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']
)
```

```
Found 328 images belonging to 7 classes.
```

```
test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False,
    classes=['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']
)
```

```
Found 328 images belonging to 7 classes.
```

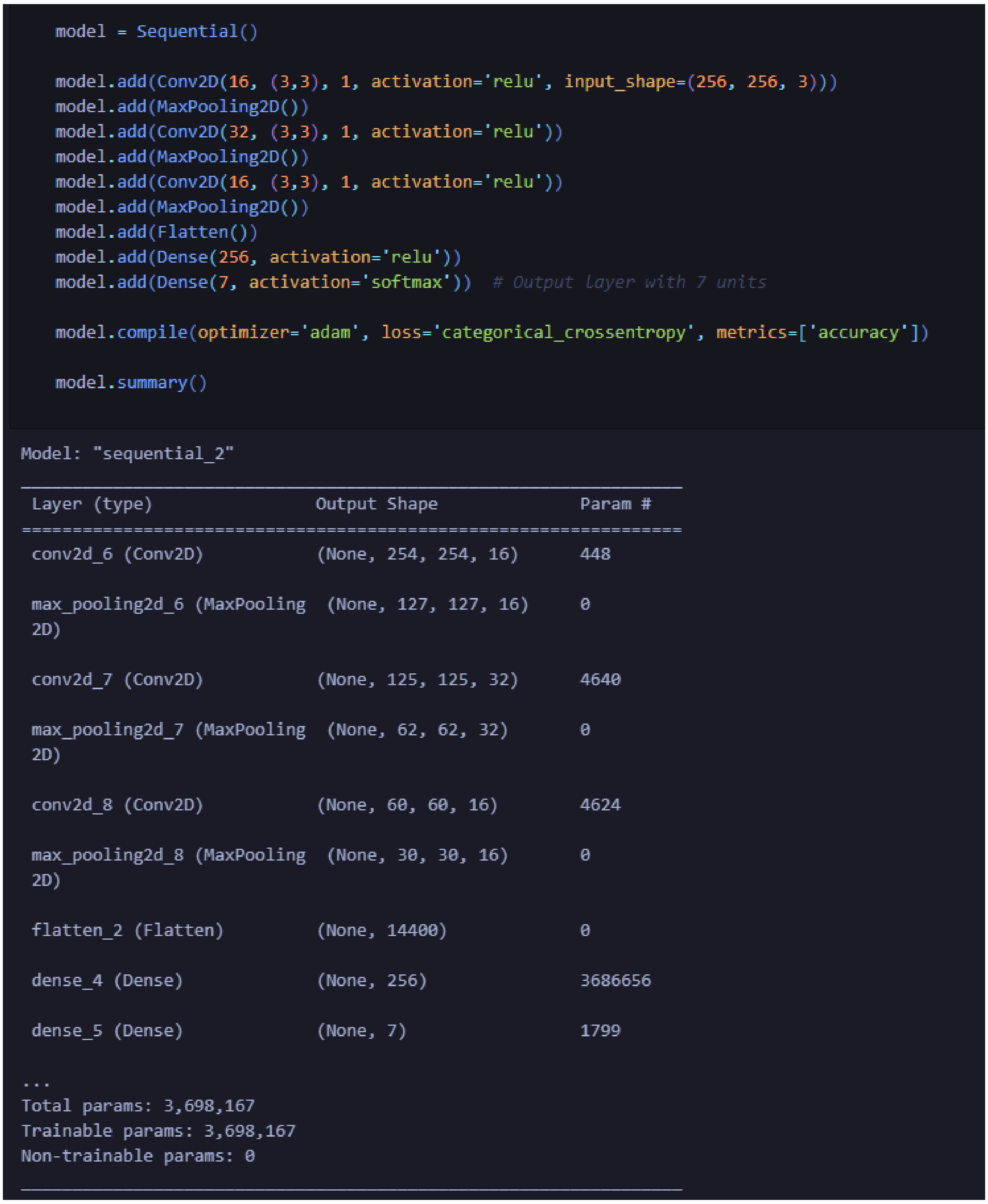
Figure 4: Data Generator Setup

Figure 5 shows the setup of the data generator for train set, test set, and validation set. These generators are configured to load and preprocess images from their respective directories, preparing the data for training and evaluation of a machine learning model.

I. ‘train_generator’: This data generator is set up to process training images from the 'train_dir' directory. It uses the ‘train_datagen’ for data augmentation and preprocessing. The key settings include resizing images to the specified dimensions of 256x256 pixels, creating batches of 32 images for each training iteration, using categorical labels for class information, shuffling the data for better randomness, and specifying the classes as 'Angry,' 'Disgust,' 'Fear,' 'Happy,' 'Neutral,' 'Sad,' and 'Surprise.' A total of 3155 images were found.

II. ‘validation_generator’: This data generator is configured for the validation dataset, located in the 'validation_dir' directory. It uses the ‘val_test_datagen’ for preprocessing. Similar to the ‘train_generator’, it resizes images, creates batches of 32, uses categorical labels, but it does not shuffle the data, ensuring that validation data remains consistent during evaluation. A total of 328 images were found.

III. ‘test_generator’: This data generator is intended for the test dataset, located in the 'test_dir' directory. Like the validation generator, it uses the ‘val_test_datagen’ for preprocessing, resizes images, creates batches of 32, uses categorical labels, and does not shuffle the data. A total of 328 images were found.



```
model = Sequential()

model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(7, activation='softmax'))  # Output layer with 7 units

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 254, 254, 16)      448

 max_pooling2d_6 (MaxPooling  (None, 127, 127, 16)     0
 2D)

 conv2d_7 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_7 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 conv2d_8 (Conv2D)           (None, 60, 60, 16)        4624

 max_pooling2d_8 (MaxPooling  (None, 30, 30, 16)       0
 2D)

 flatten_2 (Flatten)         (None, 14400)             0

 dense_4 (Dense)             (None, 256)               3686656

 dense_5 (Dense)             (None, 7)                 1799

...
Total params: 3,698,167
Trainable params: 3,698,167
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Model Implementation

The code snippet shown in Figure 5 is the implementation of the Convolutional Neural Network (CNN) model using Keras Sequential API for image classification.

I. **‘model = Sequential()’**: This initializes an empty sequential model, which allows you to stack layers in a linear sequence.

II. **‘model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256, 256, 3)))’**: This adds the first convolutional layer with 16 filters, a 3x3 kernel size, a stride of 1, and the ReLU activation function. The **‘input_shape’** parameter specifies that the input images have dimensions of 256x256 pixels with 3 color channels (RGB).

III. **‘model.add(MaxPooling2D())’**: After each convolutional layer, a max-pooling layer is added to downsample the spatial dimensions of the feature maps.

IV. The next two lines add similar layers: a convolutional layer followed by max-pooling, with 32 filters and 16 filters, respectively.

V. **‘model.add(Flatten())’**: This layer flattens the output from the previous layers into a 1D vector, preparing it for the fully connected layers.

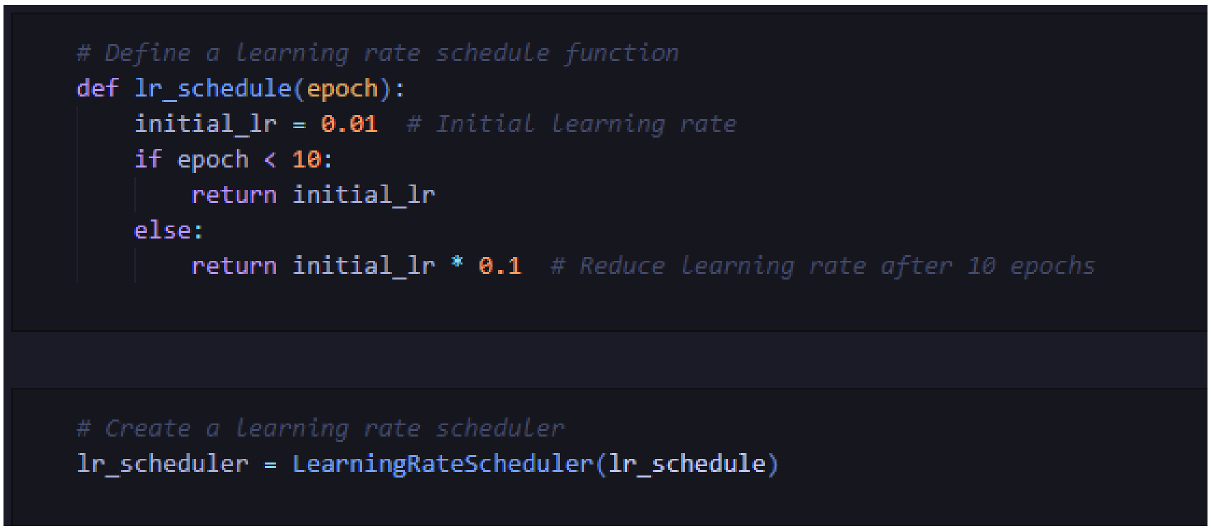
VI. **‘model.add(Dense(256, activation='relu'))’**: This adds a fully connected (dense) layer with 256 neurons and ReLU activation.

VII. **‘model.add(Dense(7, activation='softmax'))’**: The final layer is another dense layer with 7 neurons, corresponding to the 7 classes ('Angry,' 'Disgust,' 'Fear,' 'Happy,' 'Neutral,' 'Sad,' and 'Surprise') in the output. The activation function used here is softmax, which is suitable for multi-class classification tasks.

VIII. **‘model.compile(...)’**: This compiles the model, specifying the optimizer as 'adam,' the loss function as 'categorical_crossentropy' (commonly used for multi-class classification), and the metric for evaluation as 'accuracy.'

IX. **‘model.summary()’**: Finally, this command provides a summary of the model architecture, displaying the layer types, output shapes, and the number of trainable parameters in each layer.

The resulting model is a simple CNN architecture for image classification, which will be trained using the data generators and configuration previously defined in your code.



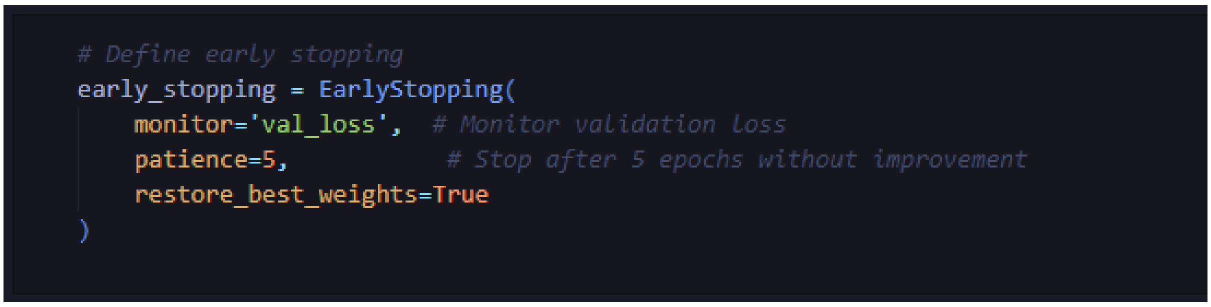
```python
# Define a learning rate schedule function
def lr_schedule(epoch):
    initial_lr = 0.01  # Initial learning rate
    if epoch < 10:
        return initial_lr
    else:
        return initial_lr * 0.1  # Reduce learning rate after 10 epochs


# Create a learning rate scheduler
lr_scheduler = LearningRateScheduler(lr_schedule)
```

Figure 6: Learning Schedule Implementation

Although the Learning Schedule was not incorporated into the actual model training, because the lower accuracy while training, it was built for testing purposes.

I. **‘def lr_schedule(epoch)’**: This is a Python function that takes the epoch as an argument. It defines a learning rate schedule, which will be used to adjust the learning rate during the training of a neural network. In this case, it starts with an initial learning rate of 0.01 and decreases to 10% (0.001) of the initial rate after the 10th epoch. This schedule is designed to gradually reduce the learning rate, which can help the model converge more effectively.

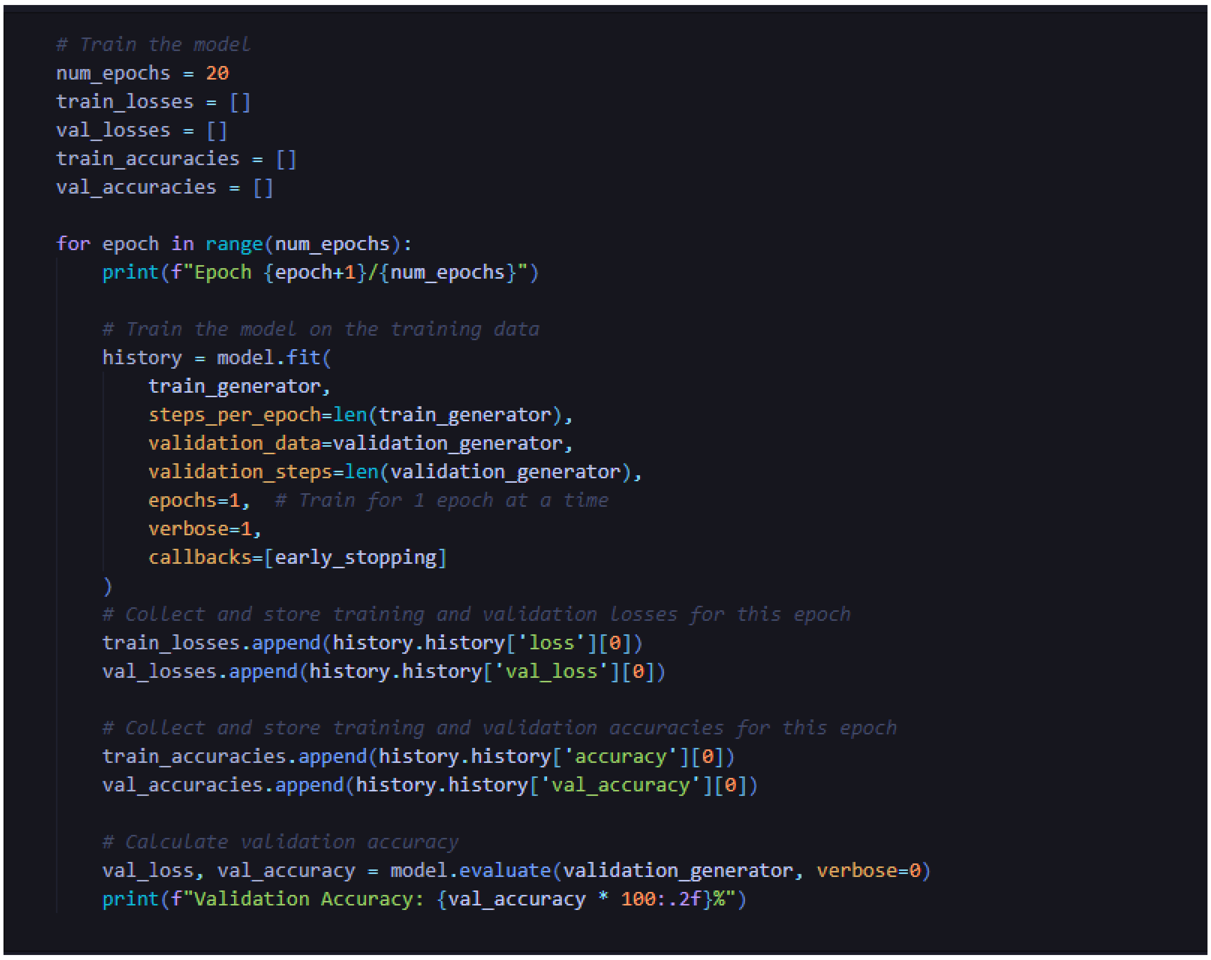
II. **‘lr_scheduler = LearningRateScheduler(lr_schedule)’**: Here, a learning rate scheduler object **‘lr_scheduler’** is created using the **‘LearningRateScheduler’** callback provided by Keras. This scheduler will apply the learning rate schedule defined in the **‘lr_schedule’** function during training. Specifically, it will adjust the learning rate based on the current epoch according to the defined schedule.



```python
# Define early stopping
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,           # Stop after 5 epochs without improvement
    restore_best_weights=True
)
```

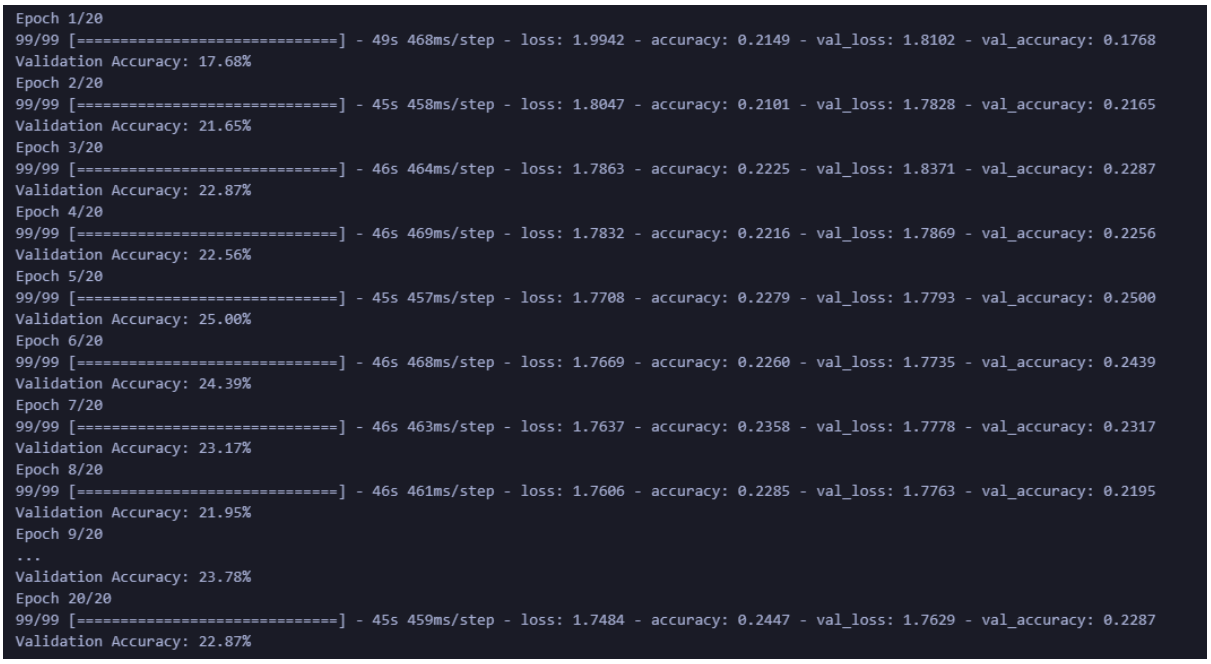
Figure 7: Early Stopping Implementation

Early Stopping is also a technique that was implemented to reduce the time to train the model. This technique is used for preventing overfitting and saving time during training by stopping the process when the model's performance on the validation set starts to degrade.



```python
# Train the model
num_epochs = 20
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

for epoch in range(num_epochs):
    print(f"Epoch {epoch+1}/{num_epochs}")

    # Train the model on the training data
    history = model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        validation_data=validation_generator,
        validation_steps=len(validation_generator),
        epochs=1,  # Train for 1 epoch at a time
        verbose=1,
        callbacks=[early_stopping]
    )
    # Collect and store training and validation losses for this epoch
    train_losses.append(history.history['loss'][0])
    val_losses.append(history.history['val_loss'][0])

    # Collect and store training and validation accuracies for this epoch
    train_accuracies.append(history.history['accuracy'][0])
    val_accuracies.append(history.history['val_accuracy'][0])

    # Calculate validation accuracy
    val_loss, val_accuracy = model.evaluate(validation_generator, verbose=0)
    print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

Figure 8: Model Training



```
Epoch 1/20
99/99 [==============================] - 49s 468ms/step - loss: 1.9942 - accuracy: 0.2149 - val_loss: 1.8102 - val_accuracy: 0.1768
Validation Accuracy: 17.68%
Epoch 2/20
99/99 [==============================] - 45s 458ms/step - loss: 1.8047 - accuracy: 0.2101 - val_loss: 1.7828 - val_accuracy: 0.2165
Validation Accuracy: 21.65%
Epoch 3/20
99/99 [==============================] - 46s 464ms/step - loss: 1.7863 - accuracy: 0.2225 - val_loss: 1.8371 - val_accuracy: 0.2287
Validation Accuracy: 22.87%
Epoch 4/20
99/99 [==============================] - 46s 469ms/step - loss: 1.7832 - accuracy: 0.2216 - val_loss: 1.7869 - val_accuracy: 0.2256
Validation Accuracy: 22.56%
Epoch 5/20
99/99 [==============================] - 45s 457ms/step - loss: 1.7708 - accuracy: 0.2279 - val_loss: 1.7793 - val_accuracy: 0.2500
Validation Accuracy: 25.00%
Epoch 6/20
99/99 [==============================] - 46s 468ms/step - loss: 1.7669 - accuracy: 0.2260 - val_loss: 1.7735 - val_accuracy: 0.2439
Validation Accuracy: 24.39%
Epoch 7/20
99/99 [==============================] - 46s 463ms/step - loss: 1.7637 - accuracy: 0.2358 - val_loss: 1.7778 - val_accuracy: 0.2317
Validation Accuracy: 23.17%
Epoch 8/20
99/99 [==============================] - 46s 461ms/step - loss: 1.7606 - accuracy: 0.2285 - val_loss: 1.7763 - val_accuracy: 0.2195
Validation Accuracy: 21.95%
Epoch 9/20
...
Validation Accuracy: 23.78%
Epoch 20/20
99/99 [==============================] - 45s 459ms/step - loss: 1.7484 - accuracy: 0.2447 - val_loss: 1.7629 - val_accuracy: 0.2287
Validation Accuracy: 22.87%
```

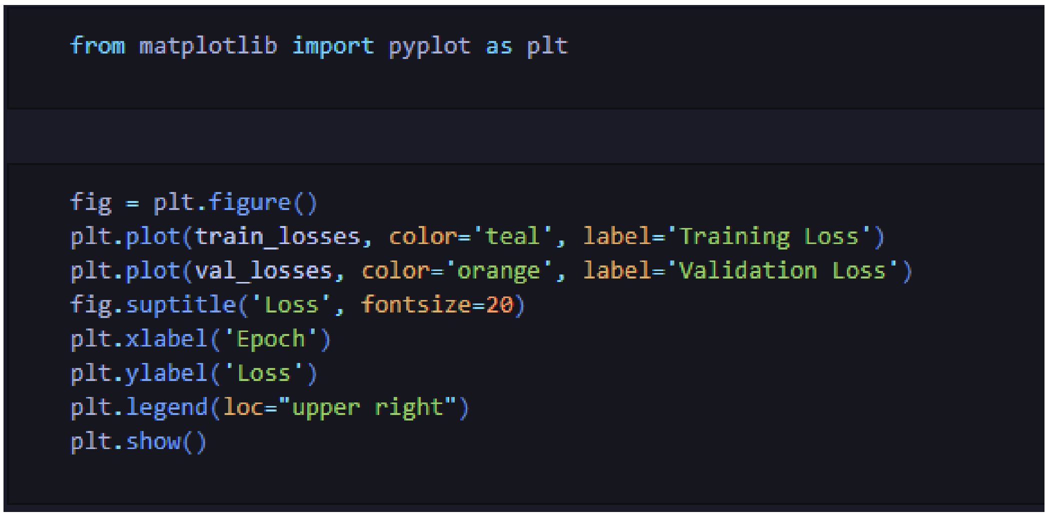
Figure 9: Model Training Result

The code snippet in Figure 8 depicted the training for the CNN model. The number of times it trains is defined by the parameter **‘num_epochs’**, which in this case is 20 times. The code also keeps track of the training losses, validation losses, training accuracies, and validation accuracies. Since the accuracy keeps going up, Early Stopping wasn’t called in to use.

The result of the training is depicted in Figure 9, which the final accuracy is 22.87%.

4.2 MODEL EVALUATIONS

The trained model exhibits promising behavior with both training and validation processes showing favorable trends.



```
from matplotlib import pyplot as plt
```

```
fig = plt.figure()
plt.plot(train_losses, color='teal', label='Training Loss')
plt.plot(val_losses, color='orange', label='Validation Loss')
fig.suptitle('Loss', fontsize=20)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc="upper right")
plt.show()
```

Figure 10: Losses Plotting Snippet

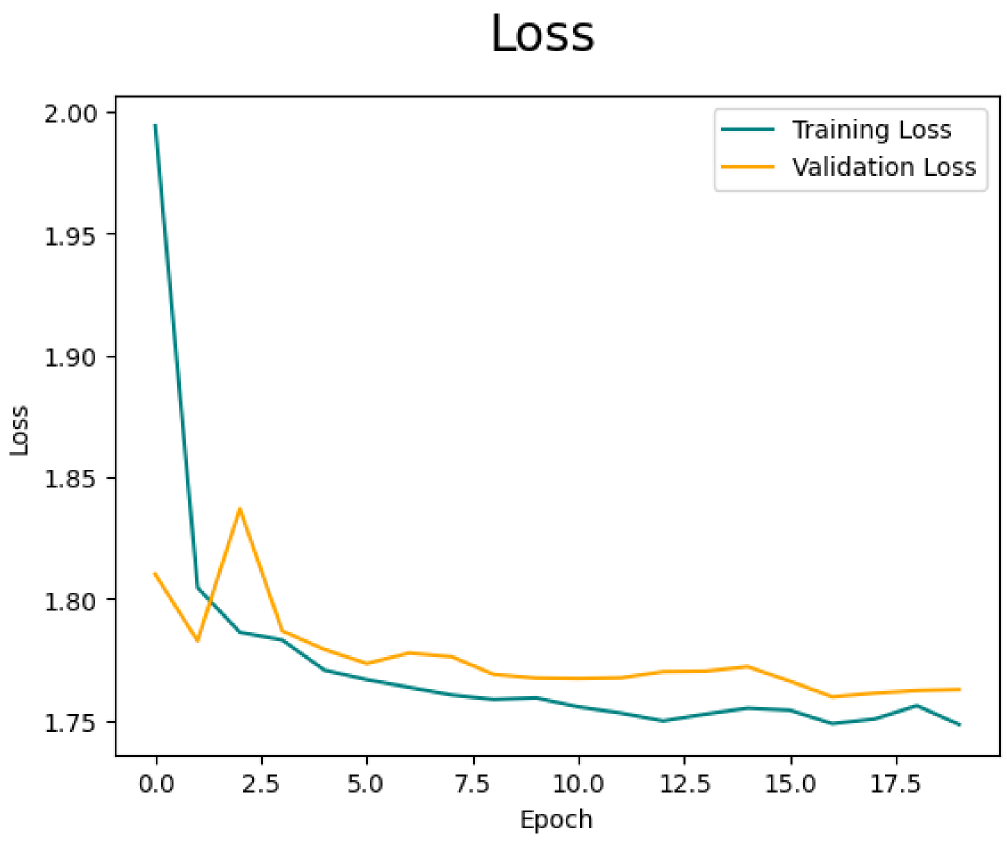
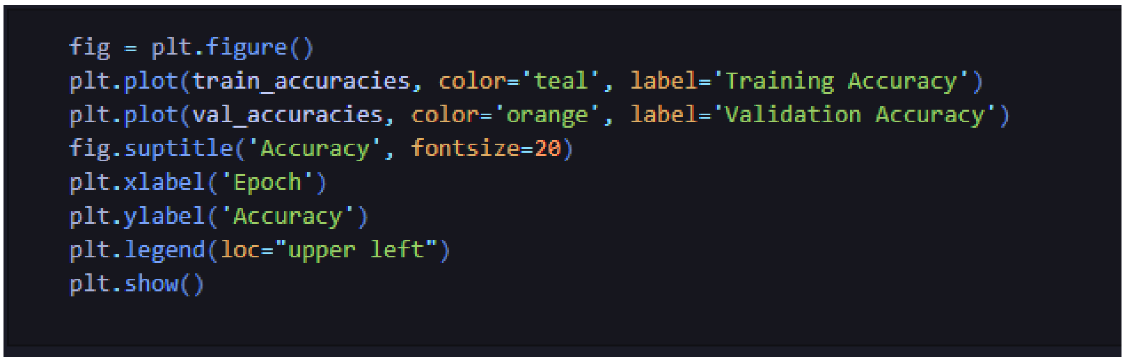


Figure 11: Loss Graph

Figure 10 and Figure 11 illustrate the loss graph. As the graph shows, the loss for both training and validating keeps going down even after 20 epochs.

The training loss consistently decreases over the course of training, demonstrating that the model is effectively learning from the training data. This reduction in training loss indicates that the model is fitting the data and adjusting its weights to minimize errors.

Similarly, the validation loss shows a gradual decrease, suggesting that the model's generalization to unseen data is improving. This is a positive sign, as it indicates that the model is not overfitting the training data.



```
fig = plt.figure()
plt.plot(train_accuracies, color='teal', label='Training Accuracy')
plt.plot(val_accuracies, color='orange', label='Validation Accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc="upper left")
plt.show()
```

Figure 12: Accuracy Plot Snippet

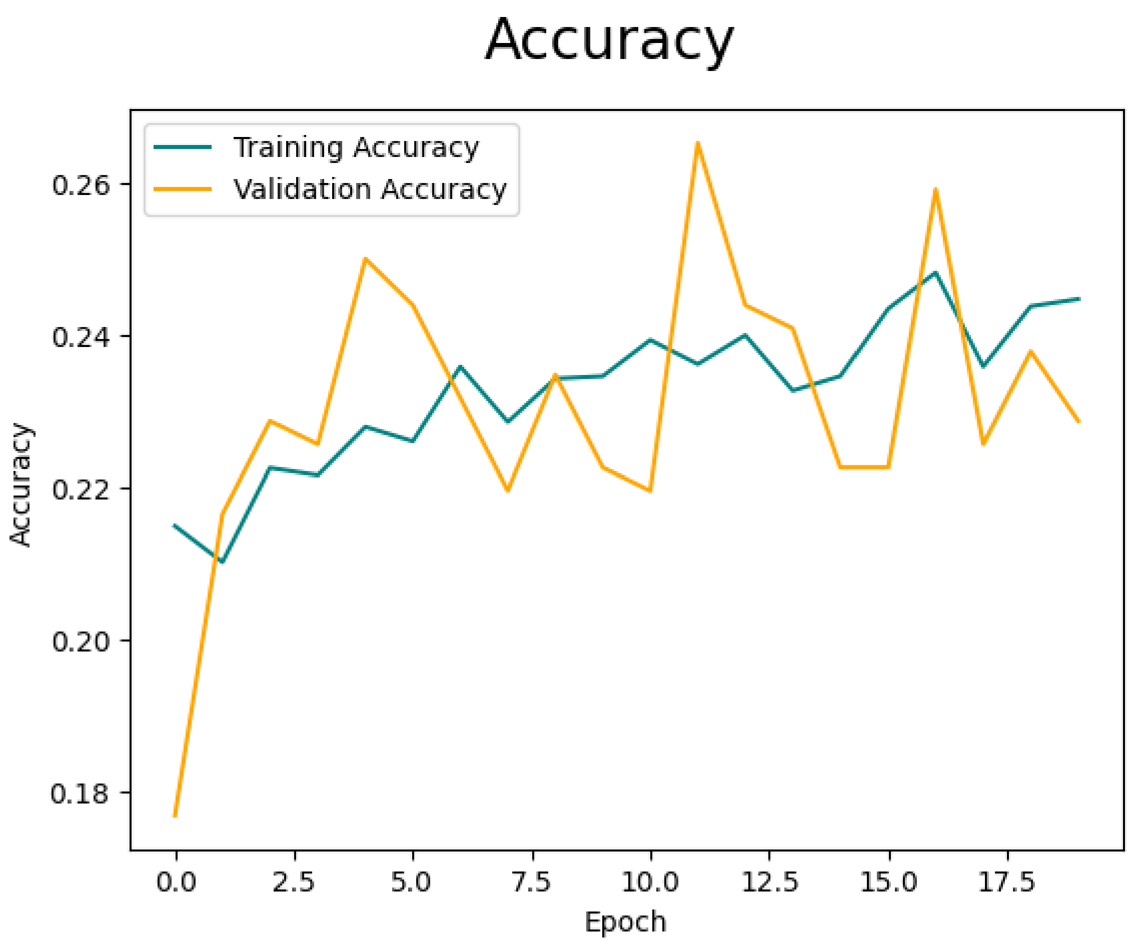


Figure 13: Accuracy Graph

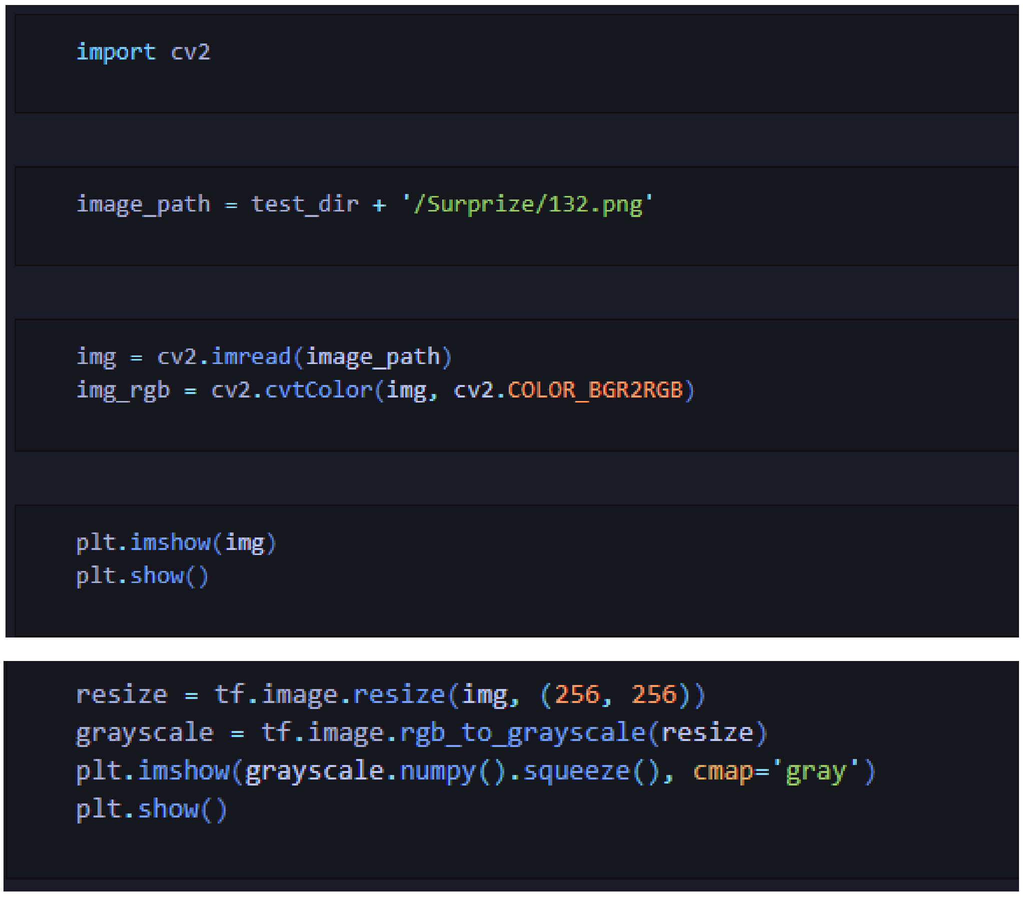
Figure 12 and Figure 13 show the accuracy of the training.

The training accuracy steadily increases with each epoch, indicating that the model is becoming more accurate in classifying the training data. This suggests that the model is capturing the underlying patterns and features in the training set.

The validation accuracy also shows improvement as training progresses. This trend is a strong indicator that the model is generalizing well to new, unseen data. The increasing validation accuracy suggests that the model is not only memorizing the training examples but is also learning to make accurate predictions on diverse data points.

Overall, this is a very good sign since the loss keeps going down and the accuracy is still going up. The observed trends in training and validation metrics are encouraging and suggest that the model is on the right path. Further refinements and optimizations can potentially yield a more robust and accurate model for the task at hand.

4.3 MODEL TESTING



```
import cv2
```

```
image_path = test_dir + '/Surprize/132.png'
```

```
img = cv2.imread(image_path)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
plt.imshow(img)
plt.show()
```

```
resize = tf.image.resize(img, (256, 256))
grayscale = tf.image.rgb_to_grayscale(resize)
plt.imshow(grayscale.numpy().squeeze(), cmap='gray')
plt.show()
```

Figure 14: Load Test Photo

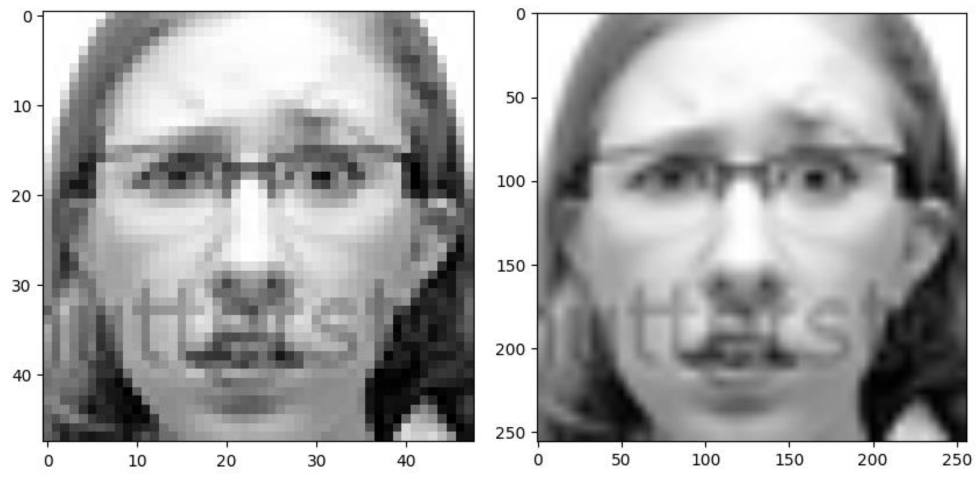
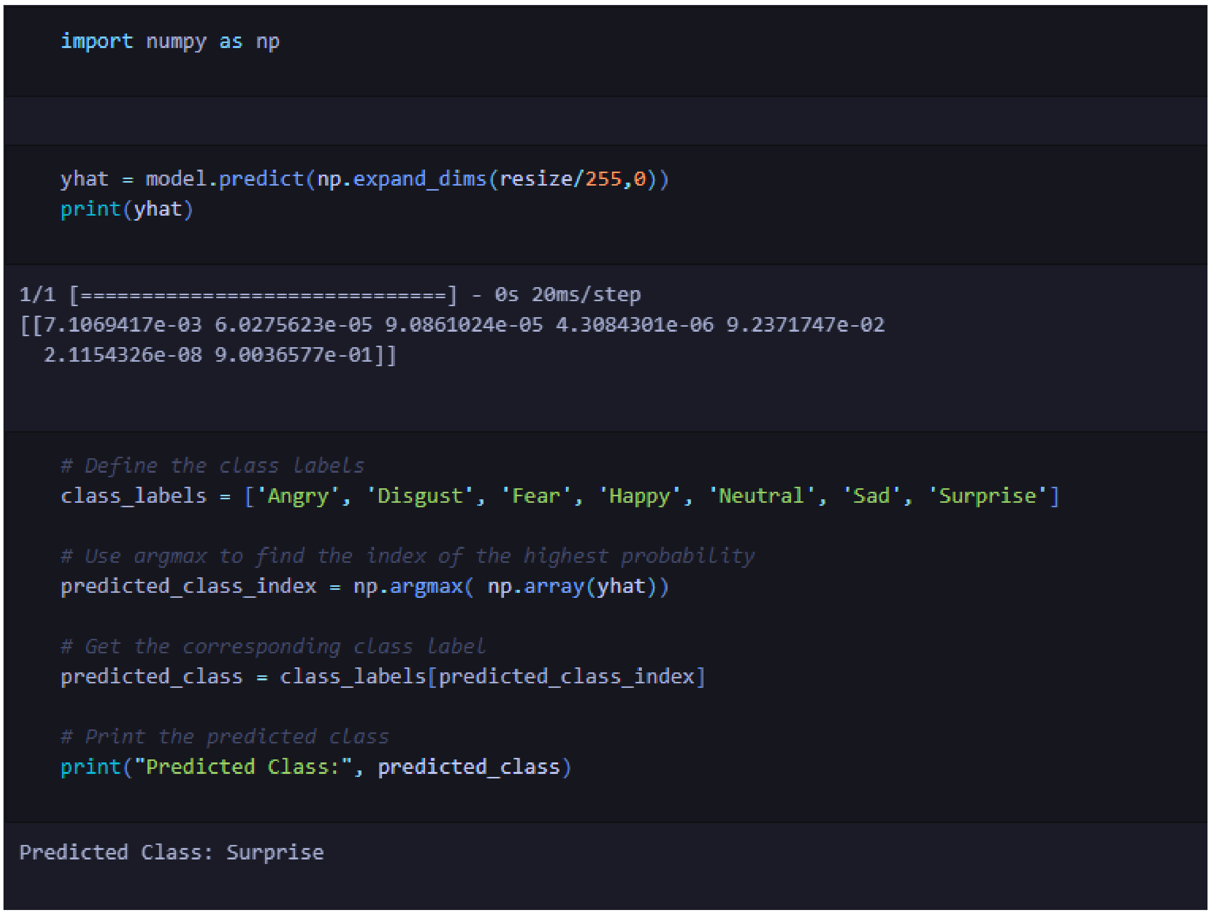


Figure 15: Test Photo

The code snippets shown in Figure 14 use **‘cv’** library to help preprocess the photo to RGB color format. It then uses the **‘tensorflow’** library to reduce the noise of the photo and turned it grayscale.



```
import numpy as np
```

```
yhat = model.predict(np.expand_dims(resize/255,0))
print(yhat)
```

```
1/1 [==============================] - 0s 20ms/step
[[7.1069417e-03 6.0275623e-05 9.0861024e-05 4.3084301e-06 9.2371747e-02
  2.1154326e-08 9.0036577e-01]]
```

```
# Define the class labels
class_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Use argmax to find the index of the highest probability
predicted_class_index = np.argmax( np.array(yhat))

# Get the corresponding class label
predicted_class = class_labels[predicted_class_index]

# Print the predicted class
print("Predicted Class:", predicted_class)
```

```
Predicted Class: Surprise
```

Figure 16: Test Result

Figure 16 shows that the code takes a preprocessed image, feeds it to the trained model, and predicts the most likely facial expression class for that image. It then prints the predicted class label.

I. **‘yhat = model.predict(np.expand_dims(resize/255, 0))’**: This line of code makes a prediction using your trained model.

II. **‘print(yhat)’**: This line prints the prediction results. **‘yhat’** likely contains the model's output, which is a probability distribution over the classes. It shows the predicted probabilities for each class.

III. **‘class_labels’**: This list defines the class labels corresponding to the different facial expressions.

IV. **‘predicted_class_index = np.argmax(np.array(yhat))’**: This line uses **‘np.argmax’** to find the index of the highest probability in the predicted probability distribution. This index corresponds to the predicted class.

V. **‘predicted_class = class_labels[predicted_class_index]’**: Using the index obtained in the previous step, this line retrieves the corresponding class label from the **‘class_labels’** list.

VI. **‘print("Predicted Class:", predicted_class)’**: Finally, this line prints the predicted class label based on the highest probability in the model’s output.\

As the output suggested, the test result came out positive and the model was able to deduce that the face in the photo was a **‘Surprise’** face.

5 IMPROVEMENTS

In our continuous pursuit of enhancing the Emotion Classifier, we have identified several key areas for improvement:

I. **Real-time Performance**: Recognizing the importance of real-time applications, we will focus on optimizing our model for low inference latency. This optimization will enable the Emotion Classifier to deliver rapid results, making it suitable for time-sensitive scenarios such as human-computer interaction and responsive user interfaces.

II. **Integration with Face Recognition**: An exciting and substantial improvement on the horizon involves the seamless integration of face recognition capabilities. This addition will empower the system not only to discern emotions accurately but also to identify individuals. By incorporating face recognition, we can offer more personalized user experiences and explore security applications where the combination of emotion and identity recognition is essential.

III. **Mental Health Disorder Classification**: As an extension of our project's scope, we plan to incorporate mental health disorder classification into our system. By leveraging advanced machine learning and deep learning techniques, we aim to identify signs of mental health disorders, such as depression and anxiety, based on facial expressions and emotional cues. This expansion holds significant promise in early detection and support for individuals experiencing mental health challenges.

By implementing these improvements, we aim to create a powerful integrated system that excels in emotion recognition, mental health disorder classification, and ethical AI principles. This system will contribute to early mental health detection, improved user experiences, and increased fairness in emotional analysis across different user groups.

REFERENCES

[1] Khairun Neesa. (2023, June 25). Depression Dataset on Facial Expression Images. Kaggle. https://www.kaggle.com/datasets/khairunneesa/depression-dataset-on-facial-ecpression-images

[2] Kiprono Elijah Koech. (2020, September 30). Softmax Activation Function: How It Actually Works. Towards Data Science. https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78

[3] Kendra Cherry. (2022, December 1). The 6 Types of Basic Emotions and Their Effect on Human Behavior. Very Well Mind. https://www.verywellmind.com/an-overview-of-the-types-of-emotions-4163976

[4] Manav Mandal. (2021, May 1). Introduction to Convolutional Neural Networks (CNN). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

[5] N.A. (N.D.). Image classification. TensorFlow API Documentation. https://www.tensorflow.org/tutorials/images/classification

[6] N.A. (N.D.). tf.all_symbols. TensorFlow API Documentation. https://www.tensorflow.org/api_docs/python/tf/all_symbols