Face Recognition and Emotion Analysis System

Image Processing Project Report

Students:

BERKAY BAKAÇ

HILASSOUN A. ROLAND SANOU

ELA TEMİZ

May 2024

The project basically consists of three parts.

1. modeling and analysis using python on jupyter notebook

2. In the android application, using the dataset we trained in the jupyter program and giving the results of face recognition and emotion analysis with the help of the camera

3. report preparation

project is not quoted.

Github is not used (there is no ready repo).

project was realized with research on the internet.

The resources we give in the "other resources" section are the resources we have researched and learned to realize our project.

an original project was created, android and notebook codes are not quoted in the project.

data set from kaggle was used.

the codes of a CNN project made on kaggle were taken to create a template and to increase the accuracy value, the CNN code was completely changed and more layers were added.

there are many analyses, graphs and classifications that were originally added in the project. the reason why the accuracy value is not higher is because there are too many classifications used in the project. in the project, both diversity was achieved by using classifications and relatively high accuracy value was achieved with the advanced cnn model by adding layers.

You can also access the code for the two pillars of the project on github:

https://github.com/Bbakac/FaceRecognitionCNN.git

https://github.com/rolandsanou/facialExpressionCNN.git

The work done with python in the Jupyter notebook, without any citations:

1. a lot of libraries, modeling, methods were added to the project.

2. originality was added to each stage and additions were made. In the code added to the pathway, a section was added to show if an error message was received.

3. Added template showing sample emotions

4. Multilayer advanced CNN model used

5. In the CNN model, the accuracy value was increased by using Conv2D, BatchNormalization, MaxPooling2D, Dropout, ReduceLROnPlateau, EarlyStopping methods.

6. 50 looping epochs were performed.

7. Achieved accuracy: test caccuracy: 0.6057397723197937

8. classification report added

9. Plot ROC curve for each class

10. pilot training history

11. def plot emotion distribution(image_label, pred_labels, emotions)

12. The complexity was checked by printing the matrix

13. plot confusion matrix with manual annotations

14. Sample data was created, a scatter plot was created showing the relationship between the actual and estimated values, a histogram Deciphering the error distribution was created

15. The error graph has been plotted. An overfitting analysis was performed.

16. Finally, the code that combines these works with android has been added:

```
"# Register your trained model
model.save('facial_recognition_model')


# Convert model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_saved_model('facial_recognition_model')
tflite_model = converter.convert()


# Save the converted model to file
with open('facial_recognition_model.tflite', 'wb') as f:
    f.write(tflite_model)”
```

Final: The app recognized the happy face 85.61 percent of the time:

the source from which we get the dataset we use in the project:

https://www.kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge/data

Only the function and the part of the tranine were taken as quotations, relevant source:

https://www.kaggle.com/code/berkaybaka/facial-expression-eda-cnn/edit

"Other resources" used for learning and research:

https://www.tensorflow.org/guide/keras
https://www.w3schools.com/python/pandas/default.asp
https://medium.com/@raguwing/face-recognition-using-cnn-architecture-in-python-f3c302c2164f
OpenCV Documentation: The official documentation of OpenCV provides detailed explanations, sample code, and usage guides. Accessible at: https://docs.opencv.org/
"Facial Expression Recognition using Convolutional Neural Networks: State of the Art" by Mollahosseini, Hossein et al. in 2016, summarizes the latest techniques in facial expression recognition employing CNNs.
"A Survey on Deep Learning Based Facial Expression Recognition" by Zhang, Zhiding et al. in 2018, investigates the impact of deep learning methods on facial expression recognition.
"Facial Expression Recognition: A Survey" by Liu, Z. et al. in 2019, reviews significant developments and methodologies in facial expression recognition.
"Facial Expression Recognition: Advances and Challenges" by Shan, C. et al. in 2014, discusses progress and hurdles in facial expression recognition.
https://www.youtube.com/watch?v=UHdrxHPRBng

Below you will find all the code for the project used in jupyter notebook and android studyo, with explanations:

Table of Contents

1. Introduction

This project offers a comprehensive face recognition and emotion analysis system utilizing advanced image processing and deep learning techniques. The system leverages Convolutional Neural Networks (CNNs) to detect and recognize faces with high accuracy. Upon detection, the system performs detailed analysis of facial expressions, recognizing emotions such as happiness, sadness, anger, surprise, and disgust. Additionally, the system incorporates advanced algorithms to handle variations in lighting conditions, facial orientations, and occlusions, ensuring robust performance in real-world scenarios. The project provides real-time feedback to users with our Android application, presenting the detected emotional states in an intuitive and user-friendly manner. This system holds potential applications in various domains, including human-computer interaction, security systems, and healthcare, contributing to the advancement of technology and enhancing user experiences.

2. Technologies Used In the Project

We used deep learning for our project and we provided the necessary environment for emotion analysis by creating an android application. Our emotion analysis system combines the strengths of deep learning, Python, and Android development to create a powerful, user-friendly system:

2.1 Deep Learning:

- **Frameworks and Libraries**: TensorFlow, Keras
- **Development Tool**: Jupyter Notebook
- **Model**: Convolutional Neural Networks (CNNs):
  CNN modeling primarily used for analyzing visual data. We used CNNs for image classification and recognition tasks, making them suitable for detecting emotions from facial expressions. Deep learning model CNNs are highly accurate in recognizing patterns and features in large datasets.

2.2 Machine Learning:

Forms the foundational principles for our deep learning models, enabling predictive accuracy and automation.

2.3 Programming Language:

Python: Python is used because it is a high-level programming language widely used for machine learning and deep learning due to its simplicity and robust libraries.

2.4 Mobile Application Development:

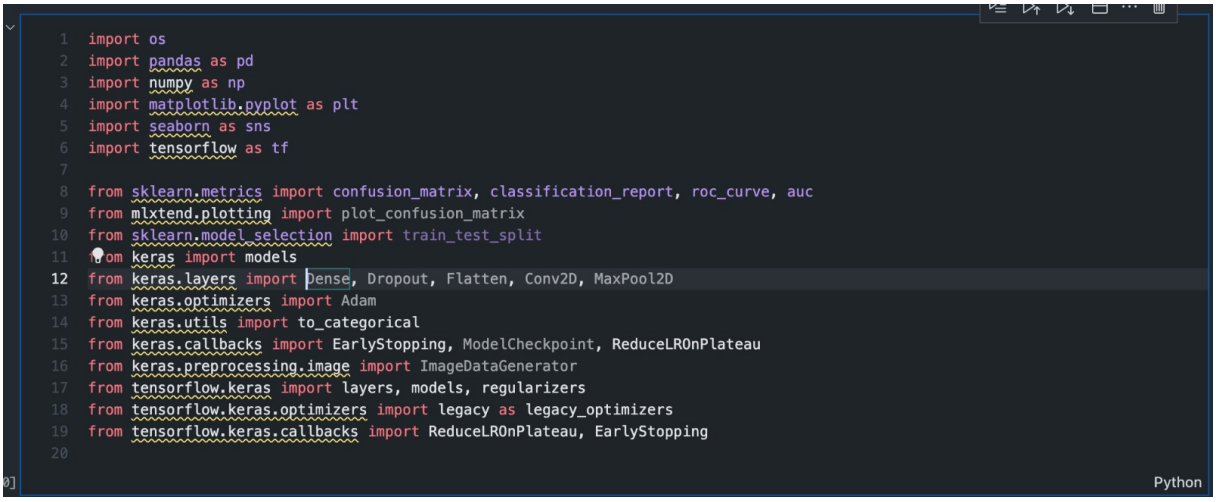
Android: Java
Android is a mobile operating system with a large user base, ensures wide accessibility and making it an excellent platform for deploying mobile applications.

These technologies work together to enable the creation of a sophisticated, efficient, and accessible emotion analysis system capable of real-time processing and analysis across multiple data types.

3. Deep Learning

3.1 Set Up the Environment

This segment sets the stage for developing an emotion analysis system by importing the necessary libraries and modules. These imports cover a wide range of functionalities including data manipulation and analysis (pandas, numpy), data visualization (matplotlib, seaborn), model building and training (TensorFlow, Keras), and performance evaluation (scikit-learn, mlxtend). This comprehensive set of tools enables the creation, training, and evaluation of complex machine learning models tailored for emotion analysis.



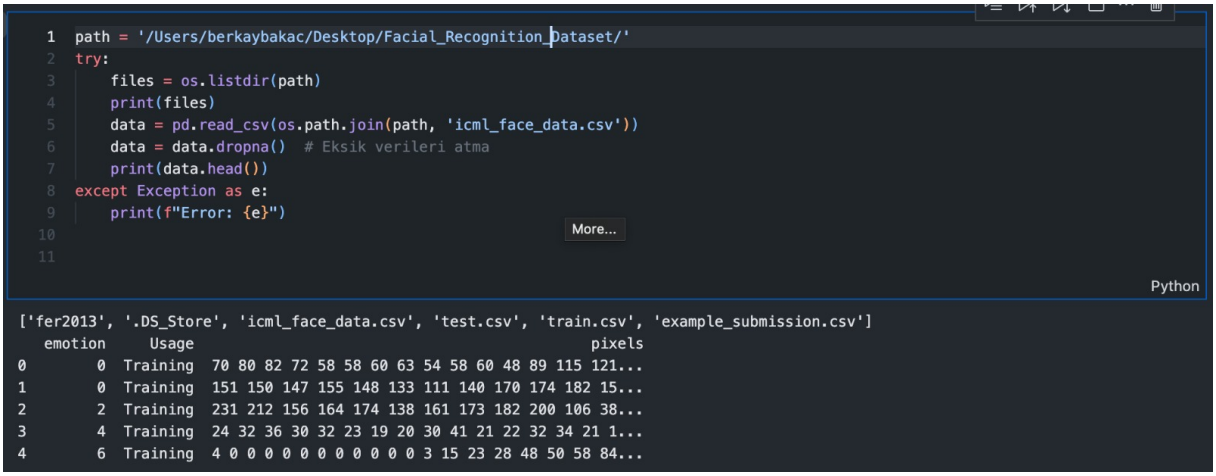
```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from mlxtend.plotting import plot_confusion_matrix
from sklearn.model_selection import train_test_split
from keras import models
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.optimizers import legacy as legacy_optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

```

Python

3.2 Setting the Path to the Dataset and Data Preview:

This code segment handles the initial stages of data loading and preprocessing for the emotion analysis system. It involves setting the path to the dataset, attempting to list the files in the specified directory, loading the data from a CSV file, and handling any potential errors that may arise during this process.

And the output verifies that the directory contains several relevant files for the project, including the facial recognition dataset. The preview of the `icml_face_data.csv` file shows its structure, with columns for emotion labels, usage type, and pixel values of the images. This data will be crucial for training and evaluating the emotion analysis system.



```python
path = '/Users/berkaybakac/Desktop/Facial_Recognition_Dataset/'
try:
    files = os.listdir(path)
    print(files)
    data = pd.read_csv(os.path.join(path, 'icml_face_data.csv'))
    data = data.dropna()  # Eksik verileri atma
    print(data.head())
except Exception as e:
    print(f"Error: {e}")

```

More...

Python

```
['fer2013', '.DS_Store', 'icml_face_data.csv', 'test.csv', 'train.csv', 'example_submission.csv']
   emotion     Usage                                             pixels
0        0  Training  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1        0  Training  151 150 147 155 148 133 111 140 170 174 182 15...
2        2  Training  231 212 156 164 174 138 161 173 182 200 106 38...
3        4  Training  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4        6  Training  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
```

3.3 `prepare_data` Function, `plot_examples` Function and `plot_all_emotions` Function:

The `prepare_data` function converts raw pixel data from a CSV file into a structured NumPy array format suitable for machine learning models, while also extracting the corresponding labels. The `plot_examples` function visualizes a few example images for a specified emotion label, aiding in understanding the dataset's content and verifying the preprocessing steps. Together, these functions facilitate data preparation and exploration, crucial steps in developing an emotion analysis system.

The `plot_all_emotions` function helps in visualizing representative images for each emotion category, providing a quick overview of the dataset.

```python
def prepare_data(data):
    """ Prepare data for modeling
        input: data frame with labels und pixel data
        output: image and label array """

    image_array = np.zeros(shape=(len(data), 48, 48))
    image_label = np.array(list(map(int, data['emotion'])))

    for i, row in enumerate(data.index):
        image = np.fromstring(data.loc[row, ' pixels'], dtype=int, sep=' ')
        image = np.reshape(image, (48, 48))
        image_array[i] = image

    return image_array, image_label

def plot_examples(label=0):
    fig, axs = plt.subplots(1, 5, figsize=(25, 12))
    fig.subplots_adjust(hspace = .2, wspace=.2)
    axs = axs.ravel()
    for i in range(5):
        idx = data[data['emotion']==label].index[i]
        axs[i].imshow(train_images[idx][:,:,0], cmap='gray')
        axs[i].set_title(emotions[train_labels[idx].argmax()])
        axs[i].set_xticklabels([])
        axs[i].set_yticklabels([])

def plot_all_emotions():
    fig, axs = plt.subplots(1, 7, figsize=(30, 12))
    fig.subplots_adjust(hspace = .2, wspace=.2)
    axs = axs.ravel()
    for i in range(7):
        idx = data[data['emotion']==i].index[i]
        axs[i].imshow(train_images[idx][:,:,0], cmap='gray')
        axs[i].set_title(emotions[train_labels[idx].argmax()])
        axs[i].set_xticklabels([])
        axs[i].set_yticklabels([])
```

3.4 `plot_image_and_emotion` Function and `plot_compare_distributions` Function:

The `plot_image_and_emotion` function compares the model's predicted probabilities with the actual emotion label for a specific test image, facilitating the evaluation of model performance. These visualizations are important for understanding the data and the effectiveness of the emotion analysis system. The `plot_compare_distributions` function facilitates the comparison of emotion prediction distributions from two different sources, such as two models or a model and the actual labels.

```python
def plot_image_and_emotion(test_image_array, test_image_label, pred_test_labels, image_number):
    """ Function to plot the image and compare the prediction results with the label """

    fig, axs = plt.subplots(1, 2, figsize=(12, 6), sharey=False)

    bar_label = emotions.values()

    axs[0].imshow(test_image_array[image_number], 'gray')
    axs[0].set_title(emotions[test_image_label[image_number]])
    axs[1].bar(bar_label, pred_test_labels[image_number], color='orange', alpha=0.7)
    axs[1].grid()

    plt.show()

def plot_compare_distributions(array1, array2, title1='', title2=''):
    df_array1 = pd.DataFrame()
    df_array2 = pd.DataFrame()
    df_array1['emotion'] = array1.argmax(axis=1)
    df_array2['emotion'] = array2.argmax(axis=1)

    fig, axs = plt.subplots(1, 2, figsize=(12, 6), sharey=False)
    x = emotions.values()

    y = df_array1['emotion'].value_counts()
    keys_missed = list(set(emotions.keys()).difference(set(y.keys())))
    for key_missed in keys_missed:
        y[key_missed] = 0
    axs[0].bar(x, y.sort_index(), color='orange')
    axs[0].set_title(title1)
    axs[0].grid()

    y = df_array2['emotion'].value_counts()
    keys_missed = list(set(emotions.keys()).difference(set(y.keys())))
    for key_missed in keys_missed:
        y[key_missed] = 0
    axs[1].bar(x, y.sort_index())
    axs[1].set_title(title2)
    axs[1].grid()

    plt.show()
```

(function) bar: Any

3.5 data[' Usage'].value_counts():

The `data[' Usage'].value_counts()` line provides a quick overview of how the dataset is split into different usage categories. This information is important for understanding the distribution of the dataset and ensuring that the training and testing phases are appropriately managed.
The `value_counts` output shows that the dataset is divided into three main parts: a large training set and two smaller test sets (public and private). This distribution ensures that the model can be trained effectively, validated during training, and evaluated rigorously after training.

```python
data[' Usage'].value_counts()
```

Python

```
 Usage
Training       28709
PublicTest      3589
PrivateTest     3589
Name: count, dtype: int64
```

3.6 dictionary `emotions`:

This mapping is useful for interpreting and labeling the emotions predicted by the model. For example, if the model predicts an output of `3`, it corresponds to the emotion 'Happy'. Similarly, a prediction of `4` corresponds to 'Sad', and so on.

```python
emotions = {0: 'Angry', 1: 'Disgust', 2: 'Fear', 3: 'Happy', 4: 'Sad', 5: 'Surprise', 6: 'Neutral'}
```

Python

3.7 Preparing Training Data, Validation Data and Testing Data:

These lines ensure that the data is appropriately split into training, validation, and testing sets, with each set consisting of image arrays and their corresponding labels, ready to be fed into machine learning or deep learning models for training and evaluation.

```python
train_image_array, train_image_label = prepare_data(data[data[' Usage']=='Training'])
val_image_array, val_image_label = prepare_data(data[data[' Usage']=='PrivateTest'])
test_image_array, test_image_label = prepare_data(data[data[' Usage']=='PublicTest'])
```

Python

3.8 Reshaping and Normalizing Training Data, Validation Data and Testing Data:

These preprocessing steps ensure that the input data is in a suitable format and scale for training and evaluating machine learning or deep learning models, which often expect pixel values to be in a normalized range for better convergence and performance.

```python
train_images = train_image_array.reshape((train_image_array.shape[0], 48, 48, 1))
train_images = train_images.astype('float32')/255
val_images = val_image_array.reshape((val_image_array.shape[0], 48, 48, 1))
val_images = val_images.astype('float32')/255
test_images = test_image_array.reshape((test_image_array.shape[0], 48, 48, 1))
test_images = test_images.astype('float32')/255
```

Python

3.9 Converting Training Labels, Validation Labels and Testing Labels:

This representation helps in training neural networks and evaluating their performance, especially when using categorical cross-entropy loss as the objective function.

```python
train_labels = to_categorical(train_image_label)
val_labels = to_categorical(val_image_label)
test_labels = to_categorical(test_image_label)
```

Python

3.10 `plot_all_emotions` Function and `plot_examples` Function:

These functions provide visualizations of example images for each emotion category, aiding in understanding and analyzing the dataset. The `plot_all_emotions` function visualizes examples for all emotion categories, while the `plot_examples` function allows for visualization of specific emotion categories.

```python
def plot_all_emotions(data, train_images, train_labels, emotions):
    fig, axs = plt.subplots(1, 7, figsize=(30, 12))
    fig.subplots_adjust(hspace=.2, wspace=.2)
    axs
    for
                                                .index[0]
                                                cmap='gray')

def plot                                        rain_labels, emotions, labels):
    fig,                                        5, figsize=(25, 12))
    fig.subplots_adjust(hspace=.2, wspace=.2)
    axs = axs.ravel()
    for label in labels:
        for i in range(5):
            idx = data[data['emotion'] == label].index[i]
            axs[label*5 + i].imshow(train_images[idx], cmap='gray')
            axs[label*5 + i].set_title(emotions[label])
            axs[label*5 + i].set_xticks([])
            axs[label*5 + i].set_yticks([])

# Tüm duyguların örneklerini görselleştirmek için
plot_all_emotions(data, train_images, train_labels, emotions)

# Belirli duyguların örneklerini görselleştirmek için
labels_to_plot = [0, 1, 2, 3, 4, 5, 6]  # Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral
plot_examples(data, train_images, train_labels, emotions, labels_to_plot)
```

```
(method) def subplots_adjust(
    left: float = ...,
    bottom: float = ...,
    right: float = ...,
    top: float = ...,
    wspace: float = ...,
    hspace: float = ...
) -> None
```

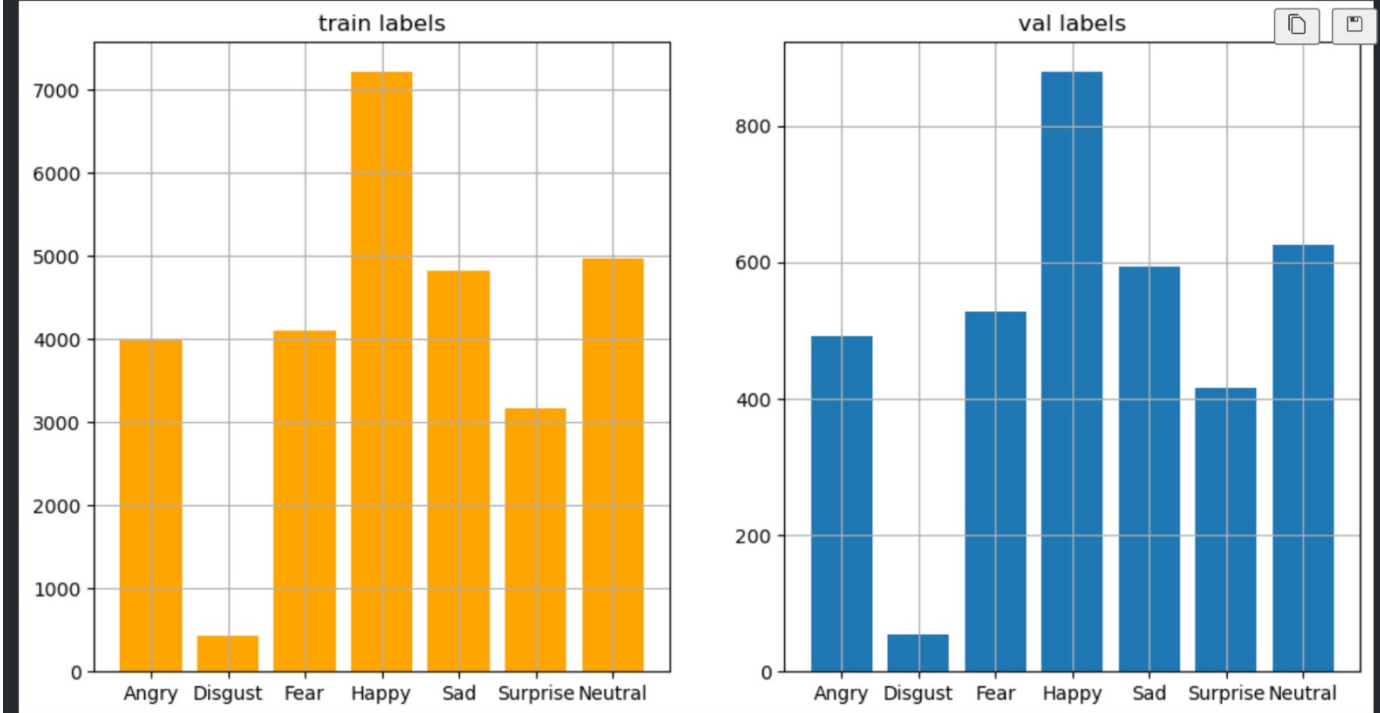
```
subplots_adjust
```

Visualizations of example images:

3.11 `plot_compare_distributions` Function:

This function call compares the distributions of emotion labels between the training and validation sets, displaying bar charts side by side with titles indicating the dataset they represent.

```
plot_compare_distributions(train_labels, val_labels, title1='train labels', title2='val labels')
```



3.12 Calculating Class Weights:

The `class_weight` dictionary contains class weights for each emotion class, calculated based on their frequency in the training dataset. These weights indicate the relative importance of each class during model training. Classes with lower frequencies are assigned higher weights to compensate for their scarcity in the dataset, ensuring that the model effectively learns from all classes.

```
class_weight = dict(zip(range(0, 7), (((data[data[' Usage']=='Training']['emotion'].value_counts()).sort_index())/len(data[data[
class_weight
```

```
{0: 0.1391549688251071,
 1: 0.01518687519593159,
 2: 0.14270786164617366,
 3: 0.2513149186666202,
 4: 0.16823992476226968,
 5: 0.11045316799609878,
 6: 0.17294228290779895}
```

3.13 `print_classification_report` Function:

This function call prints out the classification report, providing insights into the model's performance for each emotion class in terms of precision, recall, F1-score, and support.

```python
def print_classification_report(test_labels, pred_test_labels, emotions):
    """Print classification report"""
    report = classification_report(test_labels.argmax(axis=1), pred_test_labels.argmax(axis=1), target_names=emotions.values())
    print(report)

# Sınıf raporunu yazdırma
print_classification_report(test_labels, pred_test_labels, emotions)
```

Python

```
              precision    recall  f1-score   support

       Angry       0.51      0.54      0.52       467
     Disgust       0.00      0.00      0.00        56
        Fear       0.53      0.36      0.43       496
       Happy       0.79      0.85      0.82       895
         Sad       0.48      0.57      0.52       653
    Surprise       0.86      0.71      0.78       415
     Neutral       0.54      0.60      0.57       607

    accuracy                           0.62      3589
   macro avg       0.53      0.52      0.52      3589
weighted avg       0.61      0.62      0.61      3589
```

3.14 Convolutional Neural Network (CNN):

This code defines an advanced Convolutional Neural Network (CNN) model for emotion analysis.
Model architecture aims to extract hierarchical features from input images through multiple convolutional layers while mitigating overfitting with regularization techniques like dropout and batch normalization. The dense layers at the end process the extracted features and generate class predictions.

```python
# Gelişmiş CNN modeli tanımlama
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1), kernel_regularizer=regularizers.l2(0.01)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))

model.add(layers.Dense(7, activation='softmax'))

model.compile(optimizer=legacy_optimizers.Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

```

Python

Model summary describes a convolutional neural network (CNN) architecture:

```
Model: "sequential_68"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_185 (Conv2D)         (None, 46, 46, 32)        320

 batch_normalization_56 (Bat  (None, 46, 46, 32)       128
 chNormalization)

 max_pooling2d_183 (MaxPooli  (None, 23, 23, 32)       0
 ng2D)

 dropout_135 (Dropout)       (None, 23, 23, 32)        0

 conv2d_186 (Conv2D)         (None, 21, 21, 64)        18496

 batch_normalization_57 (Bat  (None, 21, 21, 64)       256
 chNormalization)

 max_pooling2d_184 (MaxPooli  (None, 10, 10, 64)       0
 ng2D)

 dropout_136 (Dropout)       (None, 10, 10, 64)        0

 conv2d_187 (Conv2D)         (None, 8, 8, 128)         73856
...
Total params: 357,255
Trainable params: 356,551
Non-trainable params: 704
```

3.15 Training, evaluation, and prediction steps for the CNN model:

This code block completes the training, evaluation, and prediction steps for the CNN model. It utilizes callbacks to adjust learning rates and prevent overfitting, trains the model on the training data with class weights, evaluates its performance on the test data, and generates predictions for further analysis. Finally, it prints the test accuracy achieved by the model.

```python
# Callback'ler
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1)

# Modeli eğitme
history = model.fit(train_images, train_labels,
                    validation_data=(val_images, val_labels),
                    class_weight=class_weight,
                    epochs=50,
                    batch_size=64,
                    callbacks=[reduce_lr, early_stopping])

# Modeli değerlendirme
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)

# Tahminler
pred_test_labels = model.predict(test_images)
```

Python

```
Epoch 1/50
449/449 [==============================] - 32s 69ms/step - loss: 3.4381 - accuracy: 0.2132 - val_loss: 4.9494 - val_accuracy: 0.1794 -
Epoch 2/50
449/449 [==============================] - 27s 60ms/step - loss: 2.2443 - accuracy: 0.2846 - val_loss: 3.2178 - val_accuracy: 0.3770 -
Epoch 3/50
449/449 [==============================] - 27s 59ms/step - loss: 1.4991 - accuracy: 0.3300 - val_loss: 2.4826 - val_accuracy: 0.4179 -
Epoch 4/50
449/449 [==============================] - 26s 59ms/step - loss: 1.0429 - accuracy: 0.3636 - val_loss: 2.1192 - val_accuracy: 0.4313 -
Epoch 5/50
```

3.16 Accuracy achieved by the model:

This code evaluates the trained model on the test dataset (`test_images` and `test_labels`) and prints the test accuracy achieved by the model.
The test accuracy of the model is approximately 0.619 (or 61.9%). This means that the model correctly classified around 61.9% of the images in the test dataset.

```python
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test caccuracy:', test_acc)
```

Python

```
113/113 [==============================] - 1s 9ms/step - loss: 1.1044 - accuracy: 0.6186
test caccuracy: 0.6185566782951355
```

3.17 Predict labels for the test images:

This code segment utilizes the trained model to predict labels for the test images (`test_images`). After execution, the variable `pred_test_labels` will contain the predicted labels for the test dataset. These predicted labels can be further analyzed to evaluate the model's performance, visualize the predictions, or compare them with the true labels to assess the model's accuracy and effectiveness in emotion analysis.

```python
pred_test_labels = model.predict(test_images)
```

Python

```
113/113 [==============================] - 1s 8ms/step
```

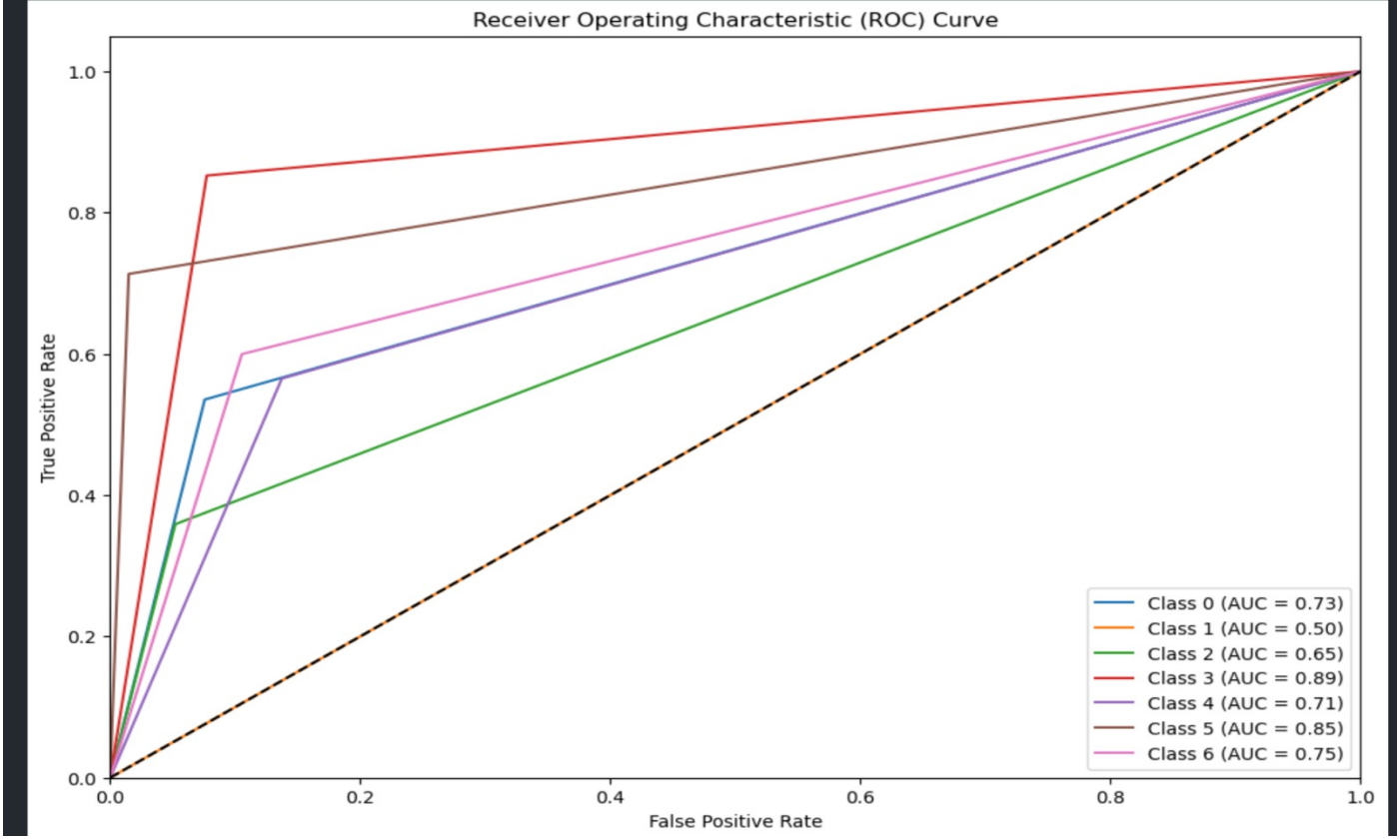
3.18 `plot_roc_curve` Function:

The provided function `plot_roc_curve` is used to plot the Receiver Operating Characteristic (ROC) curve for each class in a multi-class classification problem.

This code generates and displays the ROC curve for each emotion class, evaluating the model's performance in distinguishing between different emotions.

```python
def plot_roc_curve(test_labels, pred_test_labels, num_classes):
    """Plot ROC curve for each class"""
    fpr = {}
    tpr = {}
    roc_auc = {}

    test_labels_binarized = label_binarize(test_labels.argmax(axis=1), classes=range(num_classes))
    pred_test_labels_binarized = label_binarize(pred_test_labels.argmax(axis=1), classes=range(num_classes))

    for i in range(num_classes):
        fpr[i], tpr[i], _ = roc_curve(test_labels_binarized[:, i], pred_test_labels_binarized[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure(figsize=(12, 8))
    for i in range(num_classes):
        plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()
```

```python
# ROC eğrisi çizme
plot_roc_curve(test_labels, pred_test_labels, num_classes=len(emotions))
```



3.19 `plot_training_history` Function:

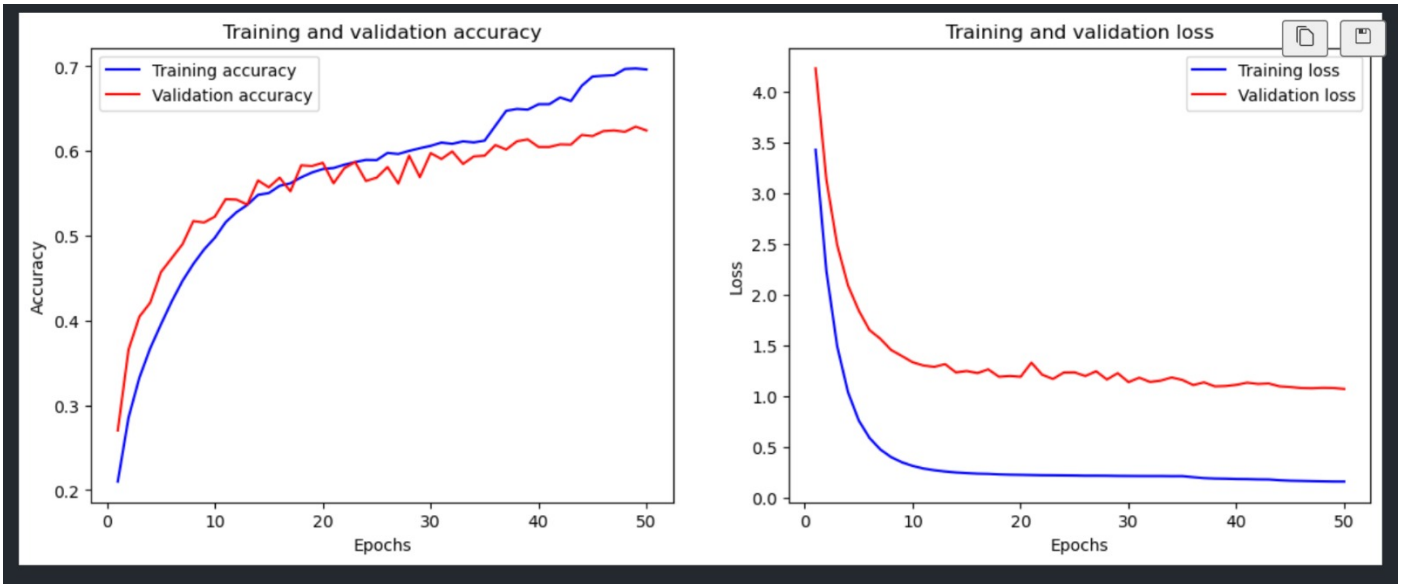
The function `plot_training_history` is used to visualize the training and validation accuracy as well as the training and validation loss over epochs.

```
plot_training_history(history)
```

This function call will visualize the training and validation accuracy as well as the training and validation loss over epochs, providing insights into the model's performance and training dynamics.

```python
def plot_training_history(history):
    """Plot training and validation accuracy and loss"""
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'b', label='Training accuracy')
    plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

# Eğitim ve doğrulama geçmişini çizme
plot_training_history(history)
```

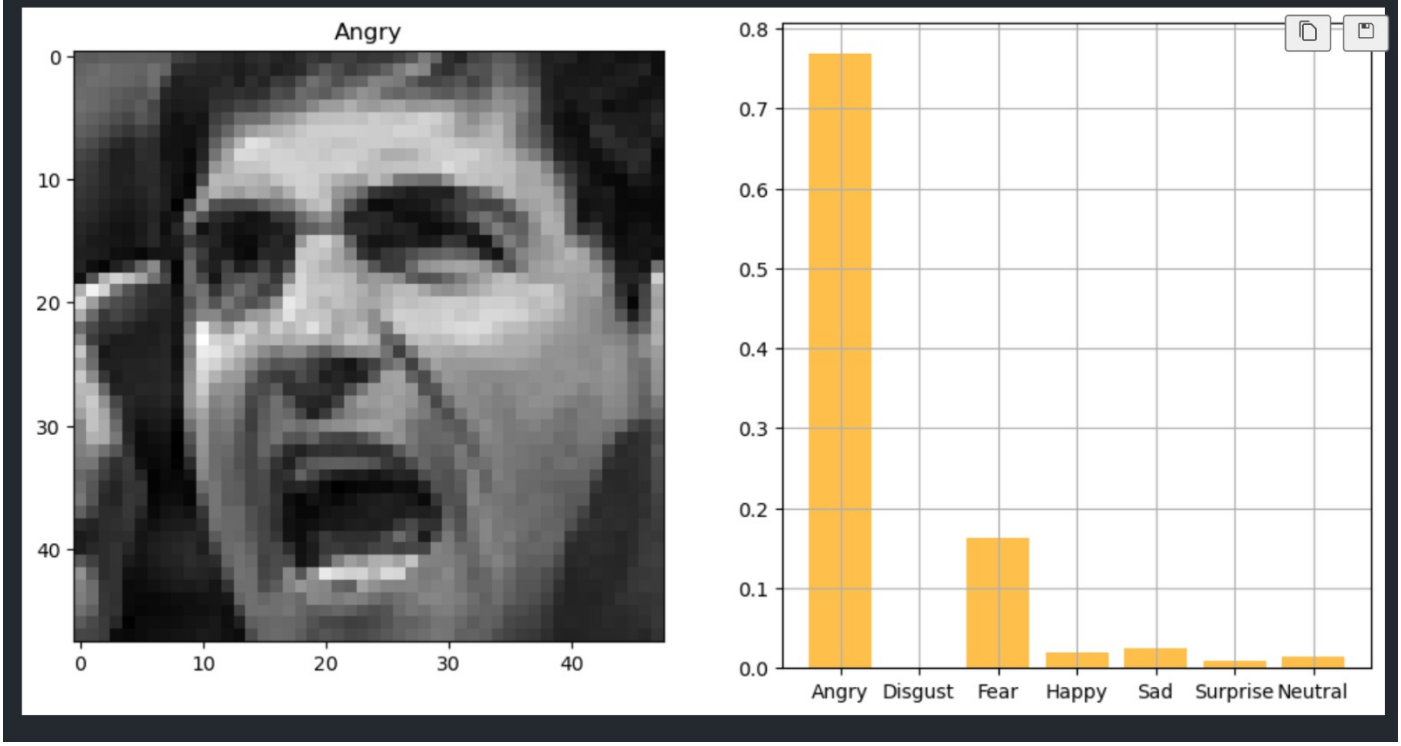
Python



3.20 `plot_image_and_emotion` Function:

This function call visualizes the image at index 151 from the test dataset along with its predicted emotion label probabilities.

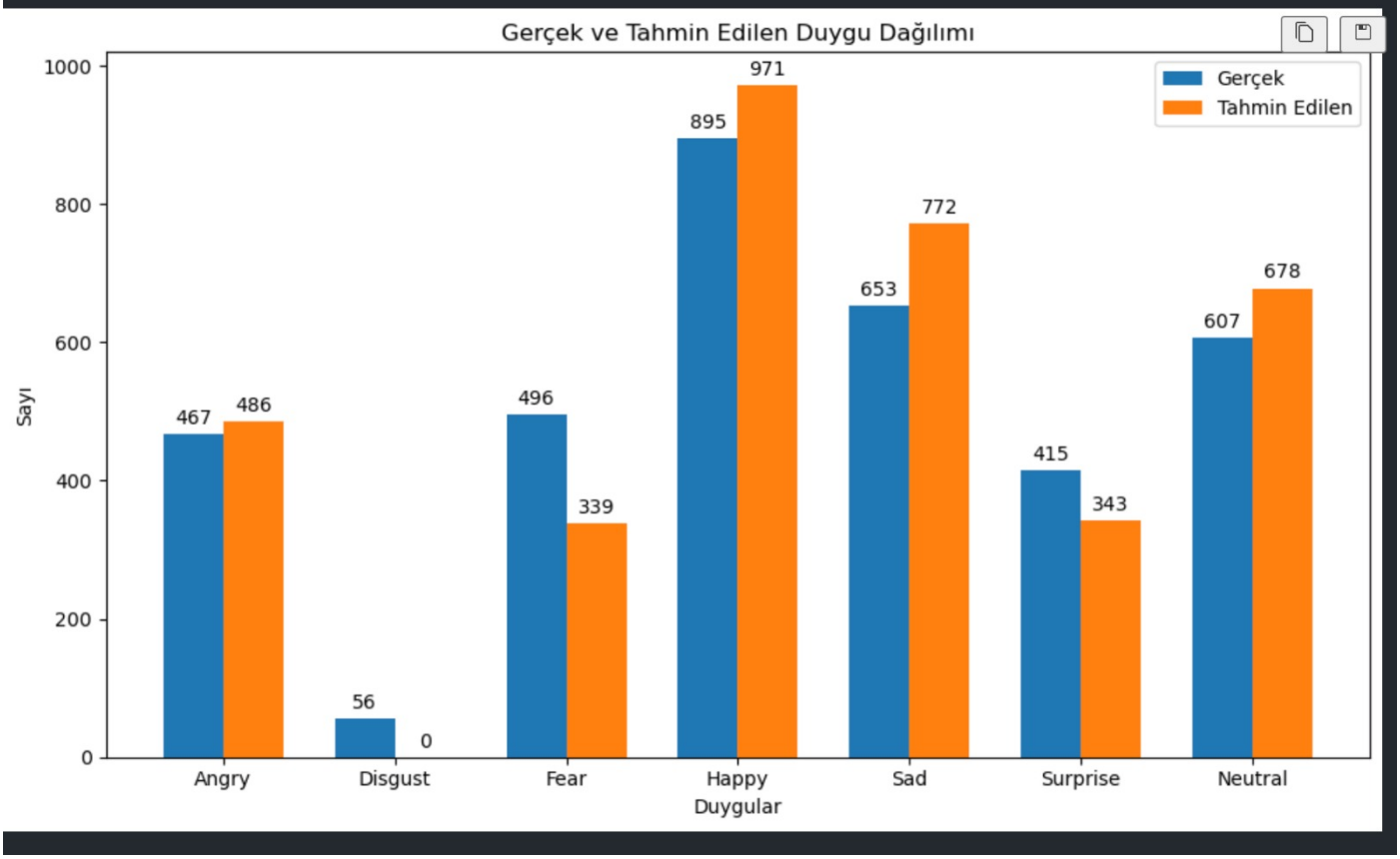
```
1  plot_image_and_emotion(test_image_array, test_image_label, pred_test_labels, 151)
```



3.21 `plot_emotion_distribution` Function:

This function call generates a bar plot illustrating the distribution of true and predicted emotions in the test dataset.

```python
# Fonksiyonu çağırma örneği
plot_emotion_distribution(test_image_label, pred_test_labels, emotions)
```



3.22 Verify the test labels, predicted labels, and the confusion matrix:

This code displays the first 10 test labels, first 10 predicted labels, and the confusion matrix, allowing us to verify the model's performance on the test dataset.

```python
# Test ve tahmin etiketlerinden ilk 10 tanesini yazdırarak kontrol edelim
print("Test etiketleri (ilk 10):", test_labels.argmax(axis=1)[:10])
print("Tahmin etiketleri (ilk 10):", pred_test_labels.argmax(axis=1)[:10])

# Karmaşıklık matrisini yazdırarak kontrol edelim
cm = confusion_matrix(test_labels.argmax(axis=1), pred_test_labels.argmax(axis=1))
print("Karmaşıklık Matrisi:\n", cm)
```

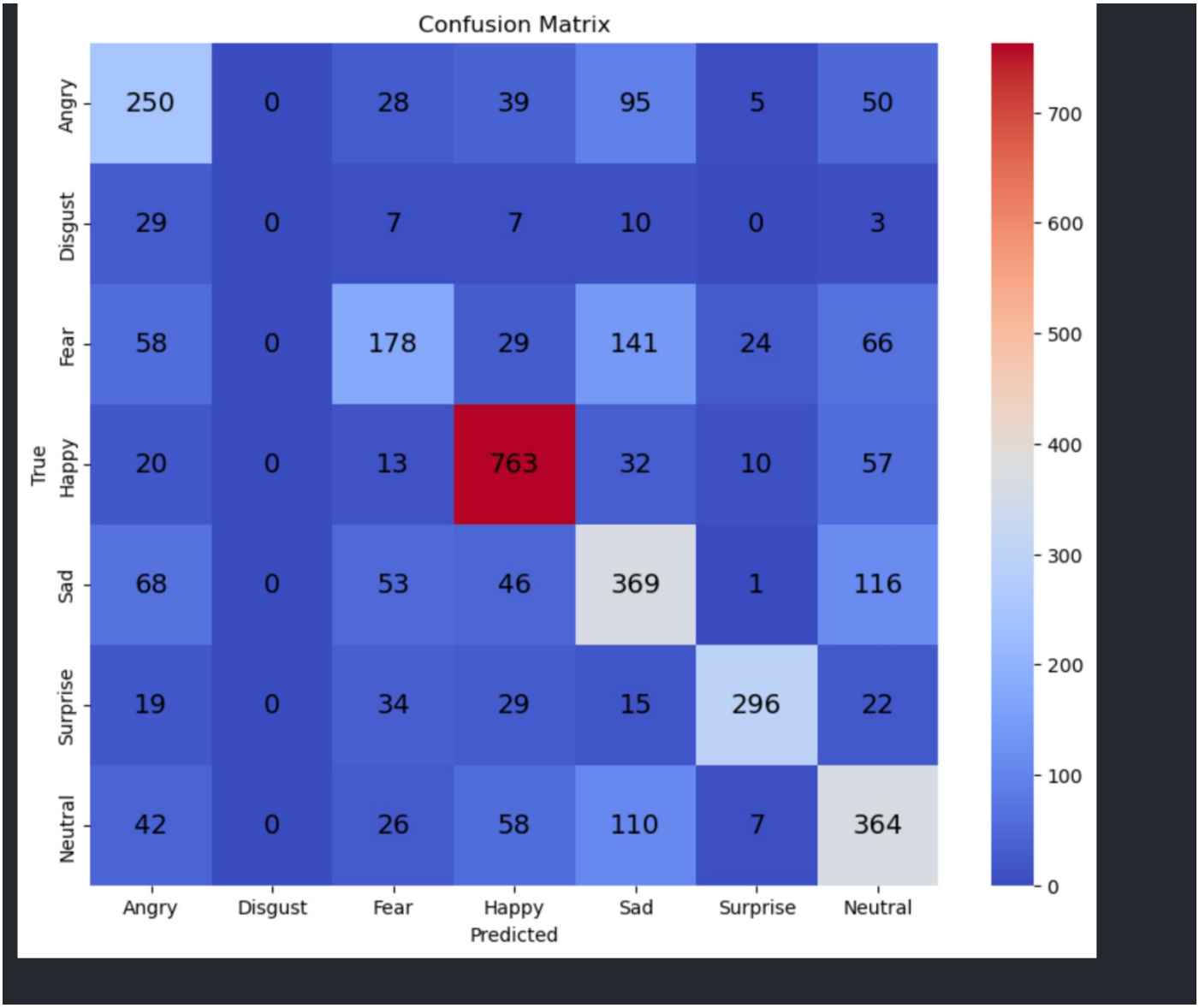
```
Test etiketleri (ilk 10): [0 1 4 6 3 3 2 0 2 0]
Tahmin etiketleri (ilk 10): [4 3 0 6 3 3 2 4 6 2]
Karmaşıklık Matrisi:
 [[250   0  28  39  95   5  50]
 [ 29   0   7   7  10   0   3]
 [ 58   0 178  29 141  24  66]
 [ 20   0  13 763  32  10  57]
 [ 68   0  53  46 369   1 116]
 [ 19   0  34  29  15 296  22]
 [ 42   0  26  58 110   7 364]]
```

3.23 `plot_confusion_matrix_with_manual_annotations` Function:

This function produces a heatmap visualization of the confusion matrix with manual annotations, helping to analyze the model's performance in predicting different emotions.

```python
def plot_confusion_matrix_with_manual_annotations(test_labels, pred_test_labels, emotions):
    """Plot confusion matrix with manual annotations"""
    cm = confusion_matrix(test_labels.argmax(axis=1), pred_test_labels.argmax(axis=1))
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=False, cmap='coolwarm', xticklabels=emotions.values(), yticklabels=emotions.values())

    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j + 0.5, i + 0.5, str(cm[i, j]), ha='center', va='center', color='black', fontsize=14)

    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()

# Karmaşıklık matrisi çizme
plot_confusion_matrix_with_manual_annotations(test_labels, pred_test_labels, emotions)
```

Pythor



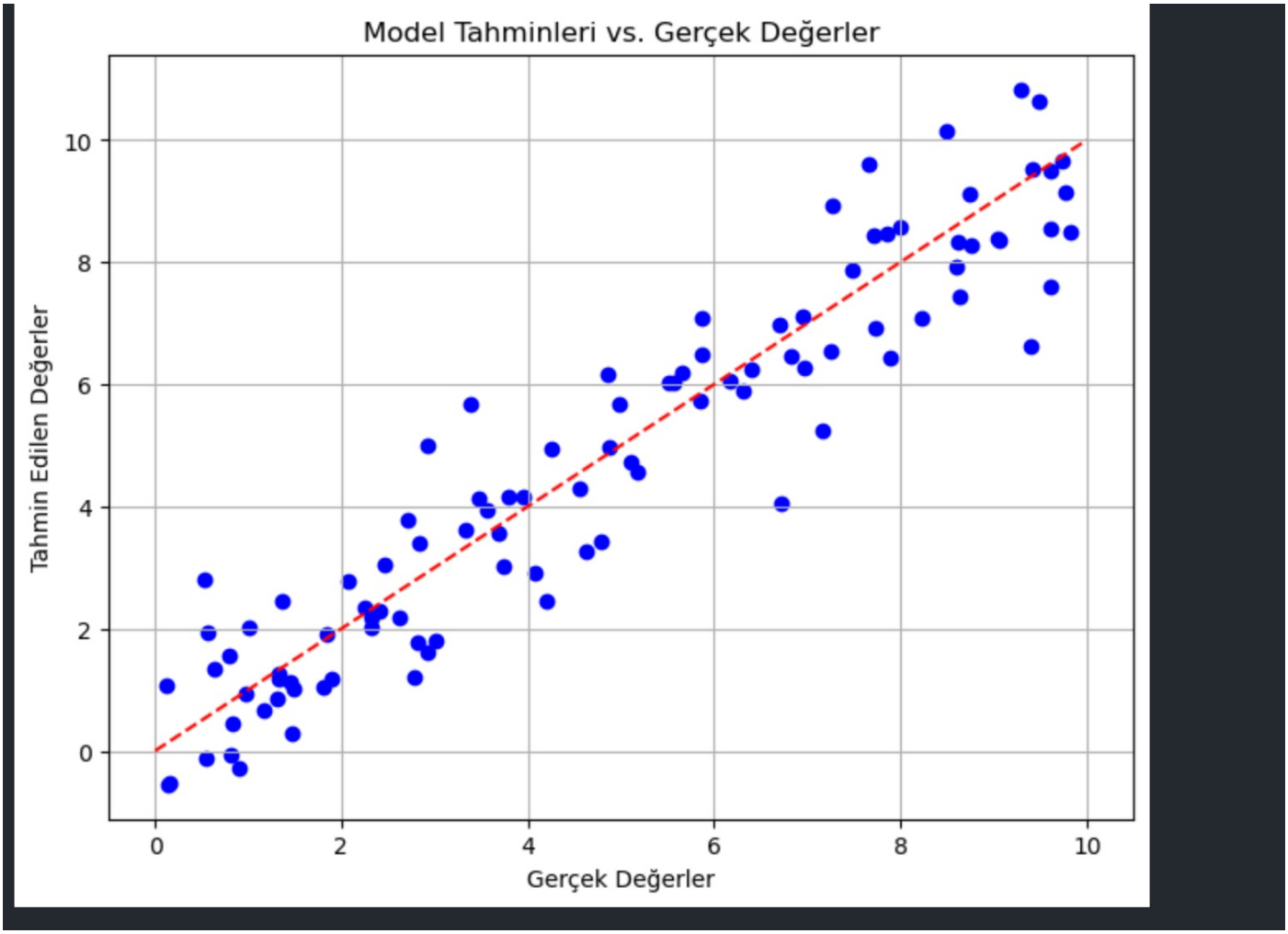
3.24 Scatter Plot of Real vs. Predicted Values:

Scatter plot is to show the relationship between real and predicted values.

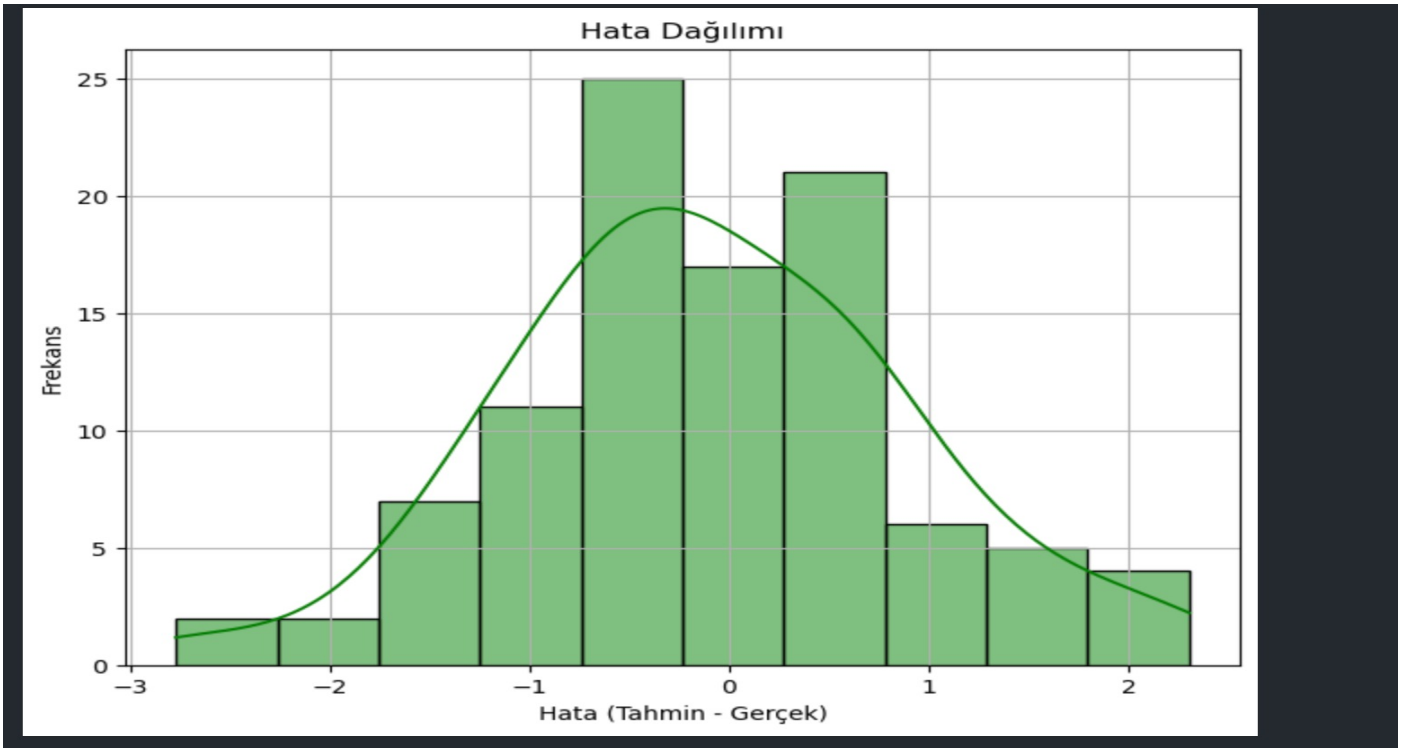
It is calculates the prediction errors by subtracting real values from predicted values and creates a histogram showing the distribution of errors.

```python
# Örnek veri oluşturalım
gercek_degerler = np.random.rand(100) * 10  # Gerçek değerler
tahmin_degerler = gercek_degerler + np.random.randn(100)  # Modelin tahmin ettiği değerler (gerçek değerlerin üzerine gürültü ek

# Gerçek ve tahmin edilen değerler arasındaki ilişkiyi gösteren bir scatter plot (dağılım grafiği) oluşturalım
plt.figure(figsize=(8, 6))
plt.scatter(gercek_degerler, tahmin_degerler, color='blue')
plt.plot([0, 10], [0, 10], color='red', linestyle='--')  # 45 derecelik doğru
plt.xlabel('Gerçek Değerler')
plt.ylabel('Tahmin Edilen Değerler')
plt.title('Model Tahminleri vs. Gerçek Değerler')
plt.grid(True)
plt.show()

# Hata dağılımını gösteren bir histogram oluşturalım
hatalar = tahmin_degerler - gercek_degerler
plt.figure(figsize=(8, 6))
sns.histplot(hatalar, kde=True, color='green')
plt.xlabel('Hata (Tahmin - Gerçek)')
plt.ylabel('Frekans')
plt.title('Hata Dağılımı')
plt.grid(True)
plt.show()
```

Python



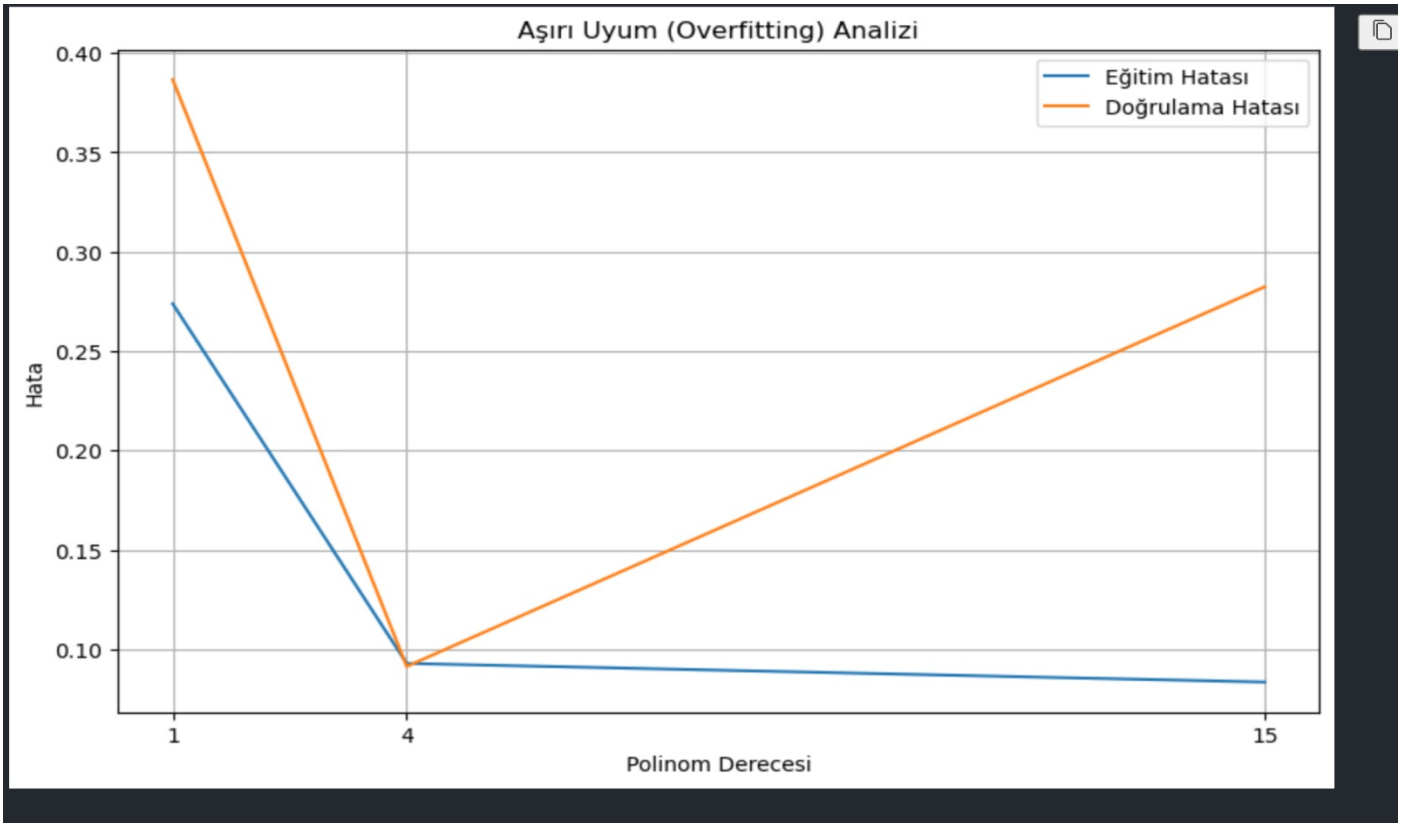
Error values are shown on the x-axis, and frequencies are shown on the y-axis.



3.25 Polynomial Regression with Different Degrees and Plotting Error Curves:

This code snippet demonstrates polynomial regression analysis for assessing overfitting. Subsequently, polynomial regression models of varying degrees (1, 4, and 15) are trained on the training data, and their respective errors on both training and validation sets are computed. These errors are then plotted against the polynomial degrees to visualize the relationship between model complexity and error. The resulting plot illustrates the training and validation errors, offering insights into the model's performance and the presence of overfitting, where higher polynomial degrees may lead to excessive fitting to the training data but poor generalization to unseen data.

```
1
2  # Örnek veri oluşturalım
3  np.random.seed(0)
4  X = 2 * np.pi * np.random.rand(100, 1)
5  y = np.sin(X) + 0.3 * np.random.randn(100, 1)
6
7  # Veriyi eğitim ve doğrulama setlerine ayıralım
8  X_train, X_val = X[:80], X[80:]
9  y_train, y_val = y[:80], y[80:]
10
11 degrees = [1, 4, 15]  # Farklı polinom dereceleri
12
13 # Her bir polinom derecesi için eğitim ve doğrulama hatasını saklayacak listeler oluşturalım
14 train_errors = []
15 val_errors = []
16
17 for degree in degrees:
18     # Modeli eğitelim
19     coeffs = np.polyfit(X_train.flatten(), y_train.flatten(), degree)
20     p = np.poly1d(coeffs)
21
22     # Eğitim ve doğrulama setlerinde hata hesaplayalım
23     train_error = np.mean((p(X_train.flatten()) - y_train.flatten()) ** 2)
24     val_error = np.mean((p(X_val.flatten()) - y_val.flatten()) ** 2)
25
26     train_errors.append(train_error)
27     val_errors.append(val_error)
28
29 # Hata grafiğini çizelim
30 plt.figure(figsize=(10, 6))
31 plt.plot(degrees, train_errors, label='Eğitim Hatası')
32 plt.plot(degrees, val_errors, label='Doğrulama Hatası')
33 plt.xlabel('Polinom Derecesi')
34 plt.ylabel('Hata')
35 plt.title('Aşırı Uyum (Overfitting) Analizi')
36 plt.xticks(degrees)
37 plt.legend()
38 plt.grid(True)
39 plt.show()
40
```



4. Deployment of the model for Android:

This code segment saves the trained facial recognition model using the TensorFlow's `save` method, storing it in a directory named 'facial_recognition_model'. Then, it converts the saved model into TensorFlow Lite format using the `TFLiteConverter` provided by TensorFlow. The converted TensorFlow Lite model is written to a file named 'facial_recognition_model.tflite' in binary mode ('wb'). This process enables the deployment of the model on resource-constrained devices such as mobile phones or microcontrollers, facilitating real-time inference tasks.

```python
# Eğitilen modelinizi kaydedin
model.save('facial_recognition_model')

# Modeli TensorFlow Lite formatına dönüştürün
converter = tf.lite.TFLiteConverter.from_saved_model('facial_recognition_model')
tflite_model = converter.convert()

# Dönüştürülen modeli dosyaya kaydedin
with open('facial_recognition_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

Python

```
2024-05-19 22:28:32.834210: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
2024-05-19 22:28:32.846309: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
2024-05-19 22:28:32.852165: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
2024-05-19 22:28:32.859603: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
2024-05-19 22:28:33.096659: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
2024-05-19 22:28:33.123204: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
2024-05-19 22:28:33.148392: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
2024-05-19 22:28:33.182988: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (th
         [[{{node inputs}}]]
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op
INFO:tensorflow:Assets written to: facial_recognition_model/assets
INFO:tensorflow:Assets written to: facial_recognition_model/assets
2024-05-19 22:28:33.971511: W tensorflow/compiler/mlir/lite/python/tf_tfl_flatbuffer_helpers.cc:364] Ignored output_format.
```

5. Mobile Application Development:

5.1 Libraries:

Andrid dependencies are external libraries or modules that our app relies on to function properly.

```
dependencies { this: DependencyHandlerScope

    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.activity)
    implementation(libs.constraintlayout)
    implementation(libs.camera.view)
    testImplementation(libs.junit)
    androidTestImplementation(libs.ext.junit)
    androidTestImplementation(libs.espresso.core)
    implementation ("org.tensorflow:tensorflow-lite:2.4.0")
    implementation ("org.tensorflow:tensorflow-lite-support:0.1.0")
    implementation ("org.tensorflow:tensorflow-lite-gpu:2.4.0") // Optional, if yo
    implementation ("androidx.camera:camera-core:1.0.0")
    implementation ("androidx.camera:camera-camera2:1.0.0")
    implementation ("androidx.camera:camera-lifecycle:1.0.0")
}
```

5.2 Project activity, permissions request and camera starting:

Permissions requests are a crucial aspect of Android development, particularly when our app requires access to sensitive data or device resources such as the camera, location, or contacts. Before our app can access these resources, it must request the necessary permissions from the user.

```
private static final String[] emotions = {"Angry", "Disgust", "Fear", "Happy", "Sad", "Surprise", "Neutral"};
roland.sanou *
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    previewView = findViewById(R.id.view_finder);
    resultTextView = findViewById(R.id.result_text);
    captureButton = findViewById(R.id.capture_button);
    selectButton = findViewById(R.id.select_button);
    // Request camera permissions
    if (ContextCompat.checkSelfPermission( context: this, android.Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED ||
            ContextCompat.checkSelfPermission( context: this,
                    android.Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {...} else {
        startCamera();
    }
    try {
        tflite = new Interpreter(loadModelFile());
    } catch (IOException e) {
        e.printStackTrace();
    }
    roland.sanou
    captureButton.setOnClickListener(new View.OnClickListener() {
        roland.sanou
        @Override
        public void onClick(View v) { captureAndRecognize(); }
    });
    selectButton.setOnClickListener(v -> {
        Intent galleryIntent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(galleryIntent, GALLERY_REQUEST_CODE);
    });
}
```

5.3 Capture image, preprocess, grayscale and send to our model for prediction:

This code snippet demonstrates how to capture an image, preprocess it, convert it to grayscale and then send it to a machine learning model for prediction in an Android app.

```java
private void captureAndRecognize() {
    if (imageCapture != null) {
        roland.sanou
        imageCapture.takePicture(ContextCompat.getMainExecutor( context: this), new ImageCapture.OnImageCapturedCallback() {
            2 usages  roland.sanou
            @Override
            public void onCaptureSuccess(@NonNull ImageProxy image) {
                Bitmap bitmap = previewView.getBitmap();
                if (bitmap != null && bitmap.getWidth() > 0 && bitmap.getHeight() > 0) {
                    ByteBuffer inputBuffer = preprocessBitmap(bitmap);
                    float[][] output = runInference(inputBuffer);
                    displayResult(output);
                } else {
                    // Handle null or invalid bitmap
                }
                image.close();
            }

            roland.sanou
            @Override
            public void onError(@NonNull ImageCaptureException exception) {
                exception.printStackTrace();
            }
        });
    }
}
```

5.4 Loading of the model and PreprocessBitmap function:

In our Android app, we leverage the TensorFlow Lite library to load our machine learning model, facilitating efficient inference on mobile devices. The TensorFlow Lite framework optimizes model execution for mobile environments, ensuring minimal latency and resource usage. As part of our image preprocessing pipeline, we implement a function called preprocessBitmap, which serves to transform our captured image into a Bitmap format and subsequently converts it to grayscale. By transforming the image into a Bitmap representation, we enable seamless integration with the Android platform, facilitating further manipulation and analysis. Moreover, by converting the image to grayscale, we simplify its representation while preserving essential visual information.

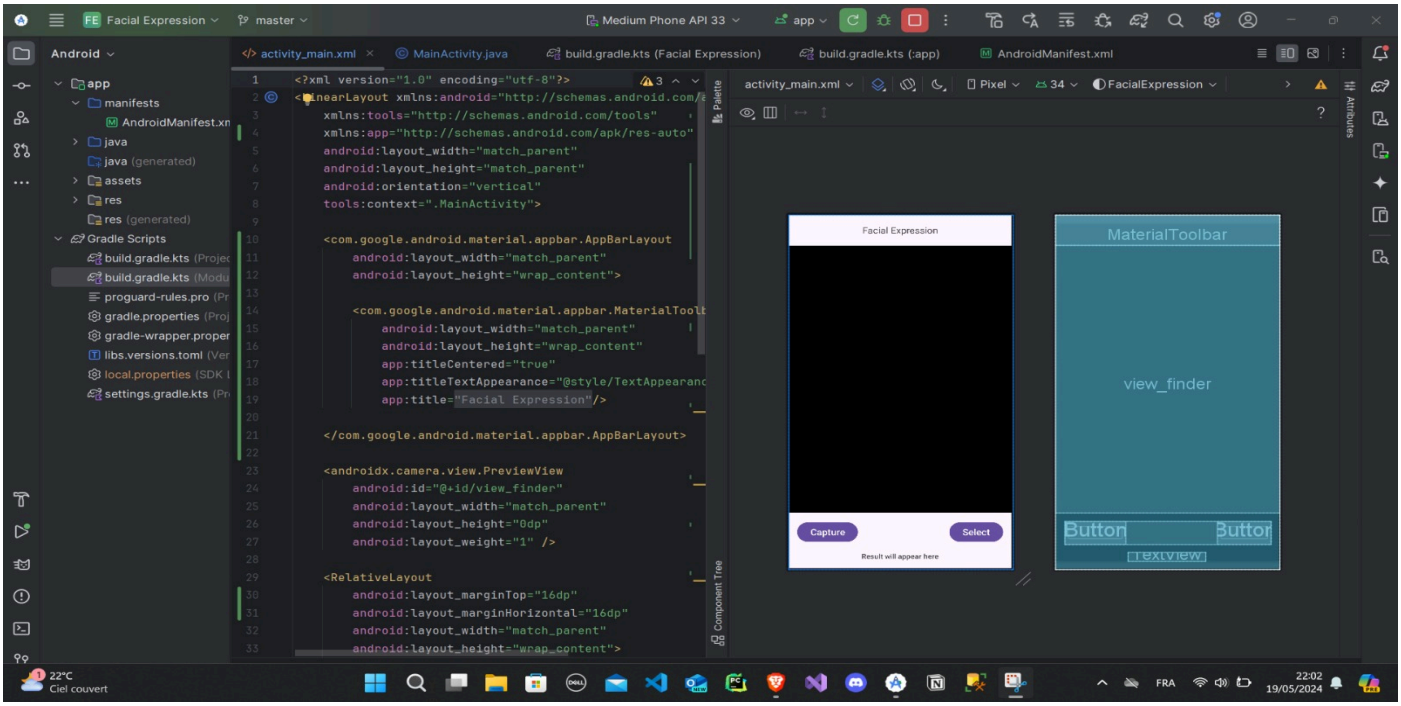
```java
1 usage  roland.sanou *
private MappedByteBuffer loadModelFile() throws IOException {
    FileInputStream fileInputStream = new FileInputStream(getAssets().openFd( fileName: "facial_recognition_model.tflite").getFileDescriptor()
    FileChannel fileChannel = fileInputStream.getChannel();
    long startOffset = getAssets().openFd( fileName: "facial_recognition_model.tflite").getStartOffset();
    long declaredLength = getAssets().openFd( fileName: "facial_recognition_model.tflite").getDeclaredLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength);
}

2 usages  roland.sanou
private ByteBuffer preprocessBitmap(Bitmap bitmap) {
    int inputSize = 48; // Adjust this to your model's input size
    ByteBuffer inputBuffer = ByteBuffer.allocateDirect( capacity: 1 * inputSize * inputSize * 1 * 4);
    inputBuffer.order(ByteOrder.nativeOrder());
    Bitmap scaledBitmap = Bitmap.createScaledBitmap(bitmap, inputSize, inputSize,  filter: true);
    int[] intValues = new int[inputSize * inputSize];
    scaledBitmap.getPixels(intValues,  offset: 0, inputSize,  x: 0,  y: 0, inputSize, inputSize);

    int pixel = 0;
    for (int i = 0; i < inputSize; i++) {
        for (int j = 0; j < inputSize; j++) {
            int value = intValues[pixel++];
            inputBuffer.putFloat( value: (value & 0xFF) / 255.0f); // Assuming grayscale
        }
    }

    return inputBuffer;
}
```

5.5 Run function and Display function:

We execute a crucial function responsible for running the preprocessed input image through our loaded machine learning model. This function orchestrates the inference process, wherein the model analyzes the input image and generates predictions regarding the emotional state conveyed within it. Following the model inference, we implement a display function designed to present the prediction results to the user in a comprehensible manner. This function computes and formats the emotion percentages inferred by the model, translating them into a user-friendly representation for display within the application's interface. These steps collectively contribute to enhancing the usability and effectiveness of our application, empowering users to interpret and respond to emotional cues captured in images with ease and precision.

```
private float[][] runInference(ByteBuffer inputBuffer) {
    float[][] output = new float[1][7]; // Adjust according to your model's output shape
    tflite.run(inputBuffer, output);
    return output;
}

2 usages  roland.sanou *
private void displayResult(float[][] output) {
    float sum = 0;
    for (float val : output[0]) {
        sum += val;
    }
    // Create an array to store percentages and their corresponding indices
    float[] percentages = new float[output[0].length];
    for (int i = 0; i < output[0].length; i++) {
        percentages[i] = (output[0][i] / sum) * 100;
    }
    // Create an array of indices to sort by percentage
    Integer[] indices = new Integer[percentages.length];
    for (int i = 0; i < indices.length; i++) {
        indices[i] = i;
    }
    // Sort the indices array based on corresponding percentage values in descending order
    Arrays.sort(indices, (a, b) -> Float.compare(percentages[b], percentages[a]));
    // Build the result string with sorted percentages
    StringBuilder result = new StringBuilder();
    for (int index : indices) {
        result.append(emotions[index])
                .append(": ")
                .append(String.format("%.2f", percentages[index]))
                .append("%\n");
    }
```

5.6 Design of our activity:


Facial Expression
master
Medium Phone API 33
app
Android
activity_main.xml
MainActivity.java
build.gradle.kts (Facial Expression)
build.gradle.kts (:app)
AndroidManifest.xml
app
manifests
AndroidManifest.xml
java
java (generated)
assets
res
res (generated)
Gradle Scripts
build.gradle.kts (Project
build.gradle.kts (Modu
proguard-rules.pro (Pr
gradle.properties (Proj
gradle-wrapper.proper
libs.versions.toml (Ver
local.properties (SDK
settings.gradle.kts (Pr
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/a
xmlns:tools="http://schemas.android.com/tools"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".MainActivity">
<com.google.android.material.appbar.AppBarLayout
android:layout_width="match_parent"
android:layout_height="wrap_content">
<com.google.android.material.appbar.MaterialToolb
android:layout_width="match_parent"
android:layout_height="wrap_content"
app:titleCentered="true"
app:titleTextAppearance="@style/TextAppearanc
app:title="Facial Expression"/>
</com.google.android.material.appbar.AppBarLayout>
<androidx.camera.view.PreviewView
android:id="@+id/view_finder"
android:layout_width="match_parent"
android:layout_height="0dp"
android:layout_weight="1" />
<RelativeLayout
android:layout_marginTop="16dp"
android:layout_marginHorizontal="16dp"
android:layout_width="match_parent"
android:layout_height="wrap_content">
Palette
Component Tree
Attributes
Pixel
34
FacialExpression
Facial Expression
Capture
Select
Result will appear here
MaterialToolbar
view_finder
Button
Buttor
TextView
22°C
Ciel couvert
FRA
22:02
19/05/2024