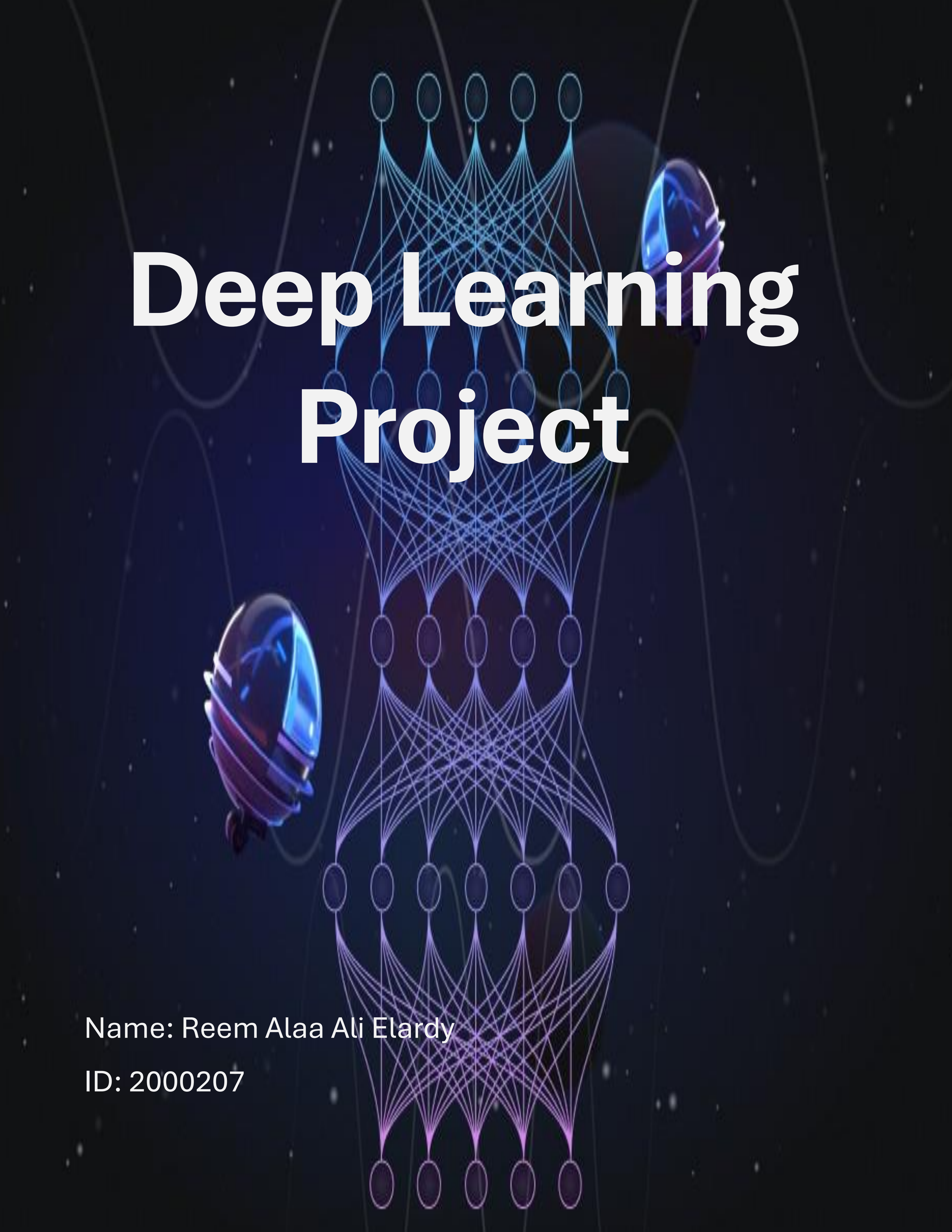


Deep Learning Project

Name: Reem Alaa Ali Elardy

ID: 2000207

Contents

1. Real-Time Face Emotion Recognition:

Face emotion recognition is an intriguing area of computer vision and artificial intelligence that focuses on identifying and interpreting human emotions through facial expressions. This technology leverages sophisticated algorithms and models to analyze facial features and predict the underlying emotional state, ranging from happiness and sadness to anger and surprise. The growing interest and advancements in this field have opened numerous applications across various domains.

2. Key Points in the project:

- **Face Detection:** The first step in the emotion recognition process is identifying and localizing faces within an image or video frame. Techniques such as the Viola-Jones algorithm, Deep Learning-based detectors, or Multi-task Cascaded Convolutional Networks (MTCNN) are commonly used.

- **Feature Extraction:** After detecting the face, the system extracts key facial features that are indicative of emotions. This is typically done using Convolutional Neural Networks (CNNs), which can learn and extract relevant features such as the shape of the mouth, the position of the eyebrows, and other facial landmarks.

- **Emotion Classification:** The extracted features are then fed into a classification model to determine the emotion. Popular models include deep learning architectures like VGGNet, ResNet, or more specialized networks trained specifically for emotion recognition.

3. Technologies and Tools

Building an effective face emotion recognition system involves a range of technologies and tools:

- Programming Languages: Python is widely used due to its robust libraries and support for machine learning and computer vision.
- Libraries and Frameworks: OpenCV for image processing, and deep learning frameworks like TensorFlow, Keras, and PyTorch for building and training models.
- Pre-trained Models and Datasets: Leveraging pre-trained models and large annotated datasets (such as FER-2013, CK+, or AffectNet) can accelerate development and improve model accuracy.

4. Project description:

The usage of pretrained models is known as transfer learning to make it easier to train the model on the specific problem its needed to, by adding or removing some layers.

So, the comparison between pretrained model to find the best for face emotion recognition is done here.

The pretrained model used:

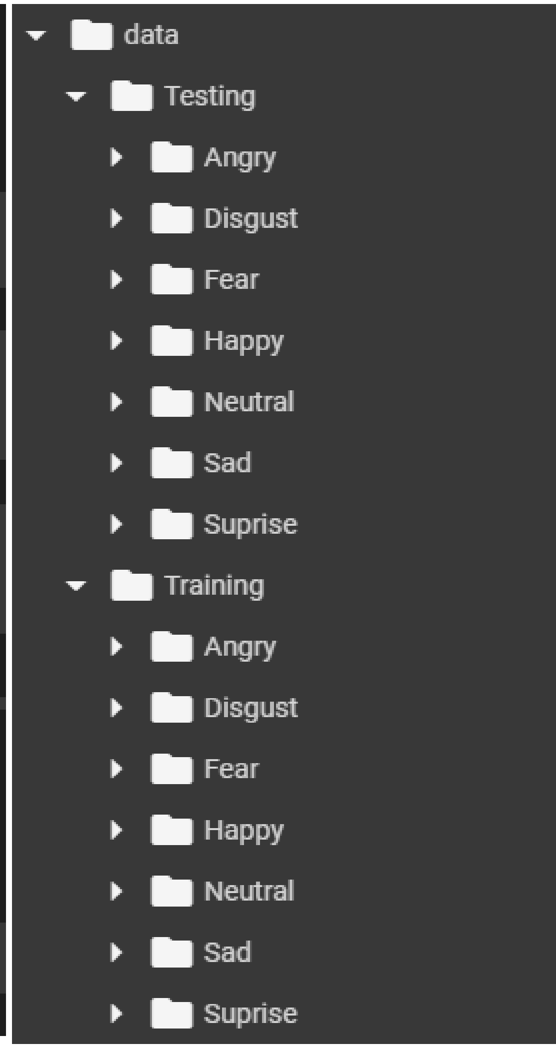
- VGG16
- VGG19
- ResNet50
- MobileNet

5. Project Steps:

A. Get the data set:

Here we get the data set from two different resources on Kaggle then combine them in one directory named as data

```
[ ] uploaded = files.upload()

    for fn in uploaded.keys():
      print('User uploaded file "{name}" with length {length} bytes'.format(
          name=fn, length=len(uploaded[fn])))

    # Then move kaggle.json into the folder where the API expects to find it.
    !mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json

    Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving kaggle.json to kaggle.json
    User uploaded file "kaggle.json" with length 66 bytes

[ ] ! kaggle datasets download -d msambare/fer2013

    Dataset URL: https://www.kaggle.com/datasets/msambare/fer2013
    License(s): DbCL-1.0
    Downloading fer2013.zip to /content
     73% 44.0M/60.3M [00:00<00:00, 142MB/s]
    100% 60.3M/60.3M [00:00<00:00, 167MB/s]

[ ] ! kaggle datasets download -d apollo2506/facial-recognition-dataset

    Dataset URL: https://www.kaggle.com/datasets/apollo2506/facial-recognition-dataset
    License(s): CC0-1.0
    Downloading facial-recognition-dataset.zip to /content
     94% 56.0M/59.7M [00:00<00:00, 63.5MB/s]
    100% 59.7M/59.7M [00:01<00:00, 60.7MB/s]

[ ] fiel1_name = '/content/facial-recognition-dataset.zip'
    file2_name = '/content/fer2013.zip'

[ ] from zipfile import ZipFile

    with ZipFile(fiel1_name,'r') as zip:
      zip.extractall()
      print('Done')

    with ZipFile(file2_name,'r') as zip:
      zip.extractall()
      print('Done')

    Done
    Done

[ ] ! mkdir data
```



B. To prevent overfitting image augmentation is done:

```
train_dir = '/content/data/Training'
test_dir = '/content/data/Testing'

def remove_checkpoints(dir_path):
    checkpoints_path = os.path.join(dir_path, '.ipynb_checkpoints')
    if os.path.exists(checkpoints_path):
        shutil.rmtree(checkpoints_path)

remove_checkpoints(train_dir)
remove_checkpoints(test_dir)

data_gen = ImageDataGenerator(
        rescale=1.0 / 255,
        rotation_range=10,  # Less rotation
        width_shift_range=0.1,  # Less shift
        height_shift_range=0.1,  # Less shift
        zoom_range=0.1,  # Less zoom
        shear_range=0.1,  # Less shear
        horizontal_flip=True,
        validation_split=0.2,
        fill_mode='nearest',
    )
```

C. Set data generation for training, validation, and testing:

```
def get_generators(target_size):
    train_gen = data_gen.flow_from_directory(
        train_dir,
        target_size=target_size,
        batch_size=16,
        class_mode='categorical',
        subset='training',
        color_mode='grayscale'
    )
    val_gen = data_gen.flow_from_directory(
        train_dir,
        target_size=target_size,
        batch_size=16,
        class_mode='categorical',
        subset='validation',
        color_mode='grayscale'
    )
    test_gen = ImageDataGenerator(rescale=1.0/255).flow_from_directory(
        test_dir,
        target_size=target_size,
        batch_size=16,
        class_mode='categorical',
        color_mode='grayscale'
    )
    return train_gen, val_gen, test_gen
```

D. Count the images in each class:

```
def count_images_in_directory(directory):
    class_counts = {}
    # Iterate over class directories and count images
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            # Count the number of files in each class directory
            class_counts[class_name] = len(os.listdir(class_path))
    return class_counts
```

E. Create the model:

Here first it take the base model which is one of the pretrained model, then the number of classes, then the input shape as our image shape can be different or the number of channels as any pretrained model take image with shape 224*224*3, then l2_reg which is the regularization and droupout_rate as the model have faced a lot of overfitting so this is was the best way to prevent it.

First add Conv2D layer as my training and testing data is in grayscale so this change it into RGB image

Then use the base model, then add layer that reduces the spatial dimensions to a single vector per feature map.

Then add a fully connected layer with L2 regularization and a dropout layer for preventing overfitting.

Then finally the final output layer has num_classes units with SoftMax activation, suitable for a multi-class classification problem.

```
def create_model(base_model, num_classes, input_shape, l2_reg=0.01, dropout_rate=0.5):
    # Create a new input layer with the grayscale input shape
    inputs = Input(shape=input_shape)

    # Add a Conv2D layer to convert grayscale input to 3-channel
    x = Conv2D(3, (3, 3), padding='same')(inputs)

    # Use the base model with this new input layer
    x = base_model(x)

    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu', kernel_regularizer=l2(l2_reg))(x)
    x = Dropout(dropout_rate)(x)
    predictions = Dense(num_classes, activation='softmax', kernel_regularizer=l2(l2_reg))(x)

    model = Model(inputs=inputs, outputs=predictions)
    return model
```

F. Train and evaluate function with call backs:

Defines callbacks for early stopping and learning rate reduction to improve model training:

- EarlyStopping: Stops training when the validation loss stops improving.
- ReduceLROnPlateau: Reduces the learning rate when the validation loss plateaus.

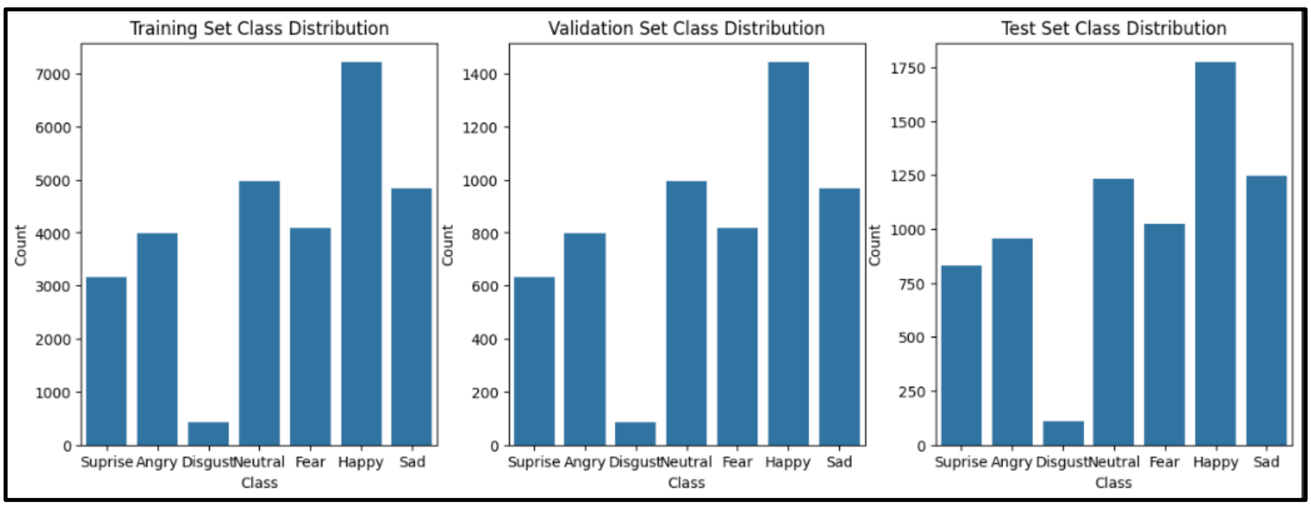
Training and Evaluation Function (train_and_evaluate):

- Trains the model using the provided training and validation data generators.
- Plots training and validation accuracy and loss over epochs to visualize performance.
- Saves the trained model to a file.
- Returns the training history for further analysis.

```
# Define callbacks for early stopping and learning rate reduction
callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5)
]

# Define a function to train, evaluate, and plot model metrics
def train_and_evaluate(model, train_gen, val_gen, test_gen, modelName, epochs=15):
    # Display the model summary
    model.summary()

    # Train the model
    history = model.fit(
        train_gen,
        validation_data=val_gen,
        epochs=epochs,
        callbacks=callbacks
    )

    # Plot training and validation accuracy
    plt.figure()
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title(f'{model.name} - Training and Validation Accuracy')
    plt.legend()
    plt.show()

    # Plot loss and validation loss
    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'{modelName} - Training and Validation Loss')
    plt.legend()
    plt.show()

    # Save the model
    model.save(f'{modelName}_model.keras')
    print("Model saved as:", f'{modelName}_model.keras')

    return history
```

G. Plot the number of images in each class in training, validation and testing:

```
train_class_counts = count_images_in_directory(train_dir)

# Calculate the validation set class counts based on the split
validation_split = 0.2
val_class_counts = {key: int(value * validation_split) for key, value in train_class_counts.items()}  # Proper multiplication

# Get test set class counts
test_class_counts = count_images_in_directory(test_dir)

# Plot class distribution for training, validation, and test datasets
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

sns.barplot(x=list(train_class_counts.keys()), y=list(train_class_counts.values()), ax=axes[0])
axes[0].set_title('Training Set Class Distribution')
axes[0].set_xlabel('Class')
axes[0].set_ylabel('Count')

sns.barplot(x=list(val_class_counts.keys()), y=list(val_class_counts.values()), ax=axes[1])
axes[1].set_title('Validation Set Class Distribution')
axes[1].set_xlabel('Class')
axes[1].set_ylabel('Count')

sns.barplot(x=list(test_class_counts.keys()), y=list(test_class_counts.values()), ax=axes[2])
axes[2].set_title('Test Set Class Distribution')
axes[2].set_xlabel('Class')
axes[2].set_ylabel('Count')

plt.show()
```



H. Start Training and Testing Models

6. Models:

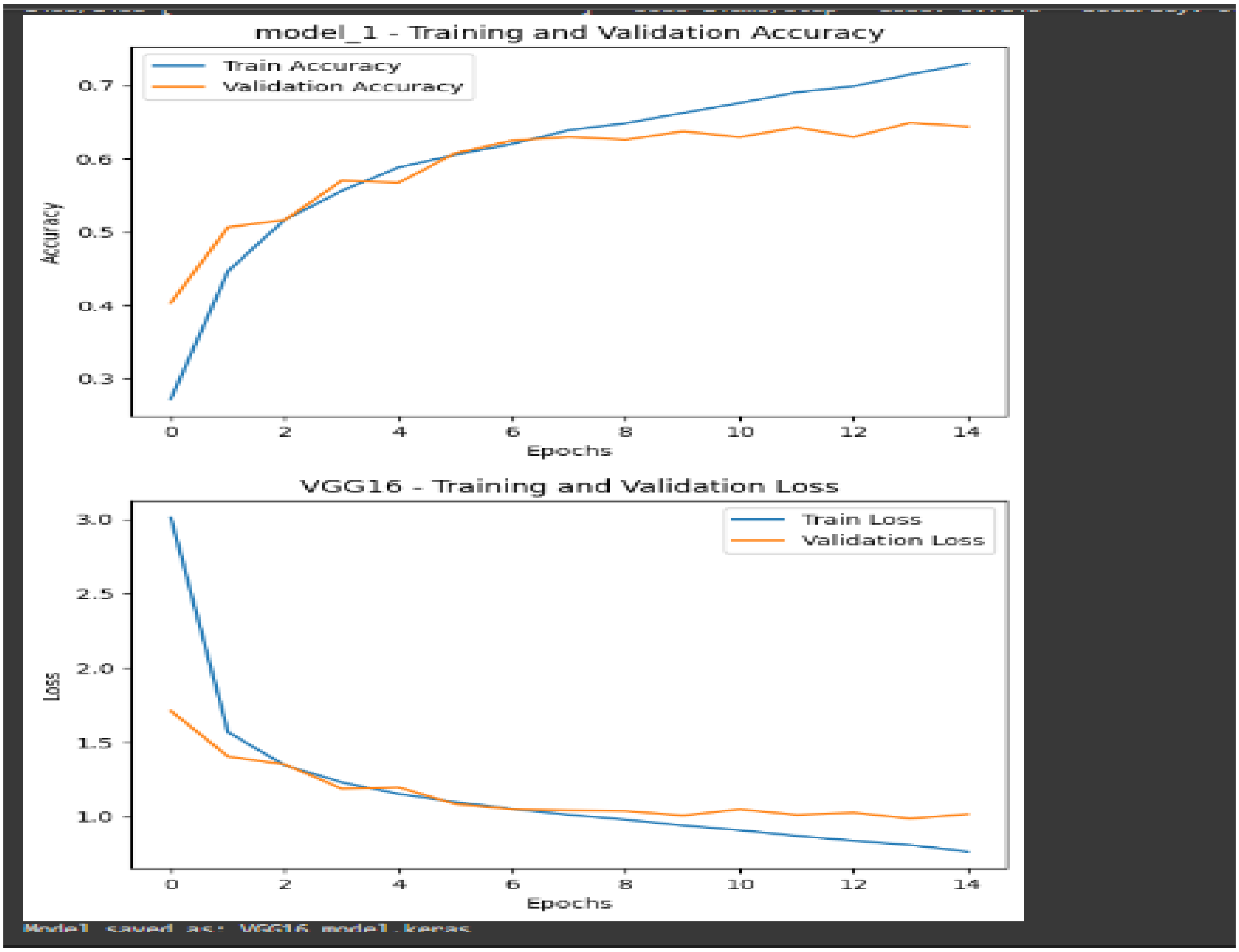
a) VGG16:

- **Creating and training the model:**

```python
from keras.optimizers import Adam
from keras.applications import VGG16, VGG19, ResNet50, ResNet152, MobileNet

train_gen, val_gen, test_gen = get_generators((224, 224))

input_shape = (224, 224, 1)

# VGG16
VGG16_model = create_model(VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3)), 7, input_shape, l2_reg=0.01, dropout_rate=0.5)
VGG16_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
history_VGG16 = train_and_evaluate(VGG16_model, train_gen, val_gen, test_gen, "VGG16")
```

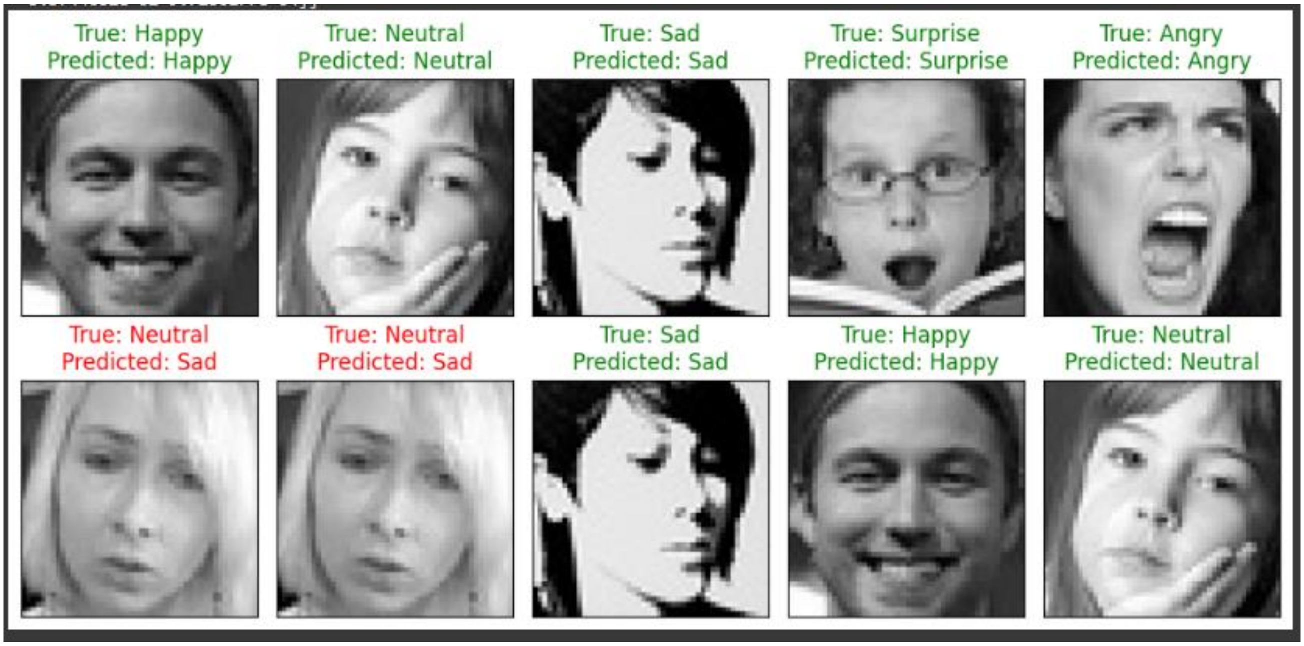
```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 1)]     0

 conv2d_1 (Conv2D)           (None, 224, 224, 3)       30

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d_1  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 1024)              525312

 dropout_1 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 7)                 7175

=================================================================
Total params: 15247205 (58.16 MB)
Trainable params: 15247205 (58.16 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
_________________________________________________________________
Epoch 1/15
1436/1436 [==============================] - 399s 259ms/step - loss: 3.0115 - accuracy: 0.2721 - val_loss: 1.7104 - val_accuracy: 0.4032 - lr: 1.0000e-04
Epoch 2/15
1436/1436 [==============================] - 353s 246ms/step - loss: 1.5671 - accuracy: 0.4464 - val_loss: 1.4033 - val_accuracy: 0.5065 - lr: 1.0000e-04
Epoch 3/15
1436/1436 [==============================] - 352s 245ms/step - loss: 1.3455 - accuracy: 0.5167 - val_loss: 1.3511 - val_accuracy: 0.5163 - lr: 1.0000e-04
Epoch 4/15
1436/1436 [==============================] - 352s 245ms/step - loss: 1.2299 - accuracy: 0.5560 - val_loss: 1.1869 - val_accuracy: 0.5701 - lr: 1.0000e-04
Epoch 5/15
1436/1436 [==============================] - 352s 245ms/step - loss: 1.1519 - accuracy: 0.5882 - val_loss: 1.1960 - val_accuracy: 0.5673 - lr: 1.0000e-04
Epoch 6/15
1436/1436 [==============================] - 352s 245ms/step - loss: 1.0974 - accuracy: 0.6054 - val_loss: 1.0841 - val_accuracy: 0.6076 - lr: 1.0000e-04
Epoch 7/15
1436/1436 [==============================] - 353s 245ms/step - loss: 1.0515 - accuracy: 0.6202 - val_loss: 1.0498 - val_accuracy: 0.6248 - lr: 1.0000e-04
Epoch 8/15
1436/1436 [==============================] - 352s 245ms/step - loss: 1.0109 - accuracy: 0.6390 - val_loss: 1.0417 - val_accuracy: 0.6299 - lr: 1.0000e-04
Epoch 9/15
1436/1436 [==============================] - 353s 246ms/step - loss: 0.9788 - accuracy: 0.6485 - val_loss: 1.0369 - val_accuracy: 0.6260 - lr: 1.0000e-04
Epoch 10/15
1436/1436 [==============================] - 352s 245ms/step - loss: 0.9398 - accuracy: 0.6625 - val_loss: 1.0061 - val_accuracy: 0.6373 - lr: 1.0000e-04
Epoch 11/15
1436/1436 [==============================] - 352s 245ms/step - loss: 0.9074 - accuracy: 0.6760 - val_loss: 1.0487 - val_accuracy: 0.6295 - lr: 1.0000e-04
Epoch 12/15
1436/1436 [==============================] - 352s 245ms/step - loss: 0.8695 - accuracy: 0.6905 - val_loss: 1.0105 - val_accuracy: 0.6427 - lr: 1.0000e-04
Epoch 13/15
1436/1436 [==============================] - 352s 245ms/step - loss: 0.8370 - accuracy: 0.6989 - val_loss: 1.0253 - val_accuracy: 0.6297 - lr: 1.0000e-04
Epoch 14/15
1436/1436 [==============================] - 374s 261ms/step - loss: 0.8079 - accuracy: 0.7152 - val_loss: 0.9858 - val_accuracy: 0.6492 - lr: 1.0000e-04
Epoch 15/15
1436/1436 [==============================] - 353s 246ms/step - loss: 0.7648 - accuracy: 0.7297 - val_loss: 1.0150 - val_accuracy: 0.6438 - lr: 1.0000e-04
```



- **Testing the model with random images in the test data:**

```
from keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# Load the saved MobileNet model
mobilenet_model = load_model("VGG16_model.keras")

# Emotion classes for the dataset
Emotion_Classes = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Assuming test_gen and model are already defined
batch_size = test_gen.batch_size

# Setting the random seed
np.random.seed()

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_gen) - 1)

# Selecting random image indices from the batch
Random_Img_Index = np.random.randint(0, batch_size, 10)

# Setting up the plot
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    # Fetching the random image and its label
    Random_Img = test_gen[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_gen[Random_batch][1][Random_Img_Index[i]], axis=0)

    # Making a prediction using the model
    Model_Prediction = np.argmax(mobilenet_model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0), axis=1)[0]
    print(mobilenet_model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0))

    # Displaying the image
    ax.imshow(Random_Img.squeeze(), cmap='gray')  # Assuming the images are grayscale
    # Setting the title with true and predicted labels, colored based on correctness
    color = "green" if Emotion_Classes[Random_Img_Label] == Emotion_Classes[Model_Prediction] else "red"
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classes[Model_Prediction]}", color=color)

plt.tight_layout()
plt.show()
```



- **Evaluate the accuracy of the model using test data:**

```python
# Evaluate on the test set and calculate additional metrics
test_loss, test_acc = VGG16_model.evaluate(test_gen)
# Get predictions and ground truth labels from the test set
predictions = VGG16_model.predict(test_gen)
predicted_classes = np.argmax(predictions, axis=1)  # Get class with highest probability
true_classes = test_gen.classes

# Calculate precision, recall, and F1-score
precision = precision_score(true_classes, predicted_classes, average='macro')
recall = recall_score(true_classes, predicted_classes, average='macro')
f1 = f1_score(true_classes, predicted_classes, average='macro')

# Print precision, recall, and F1 score
print("Accuracy: ", test_acc)
print("Loss: ", test_loss)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
449/449 [==============================] - 35s 77ms/step - loss: 1.0299 - accuracy: 0.6453
449/449 [==============================] - 29s 65ms/step
Accuracy:  0.6453050971031189
Loss:  1.029948115348816
Precision: 0.14263797426758212
Recall: 0.14418310728552836
F1 Score: 0.14023423141692598
```

- **Link to the Model:**

  https://colab.research.google.com/drive/18P50rsT_9L285Q82aPCYEGz8uDENS9bz?usp=sharing

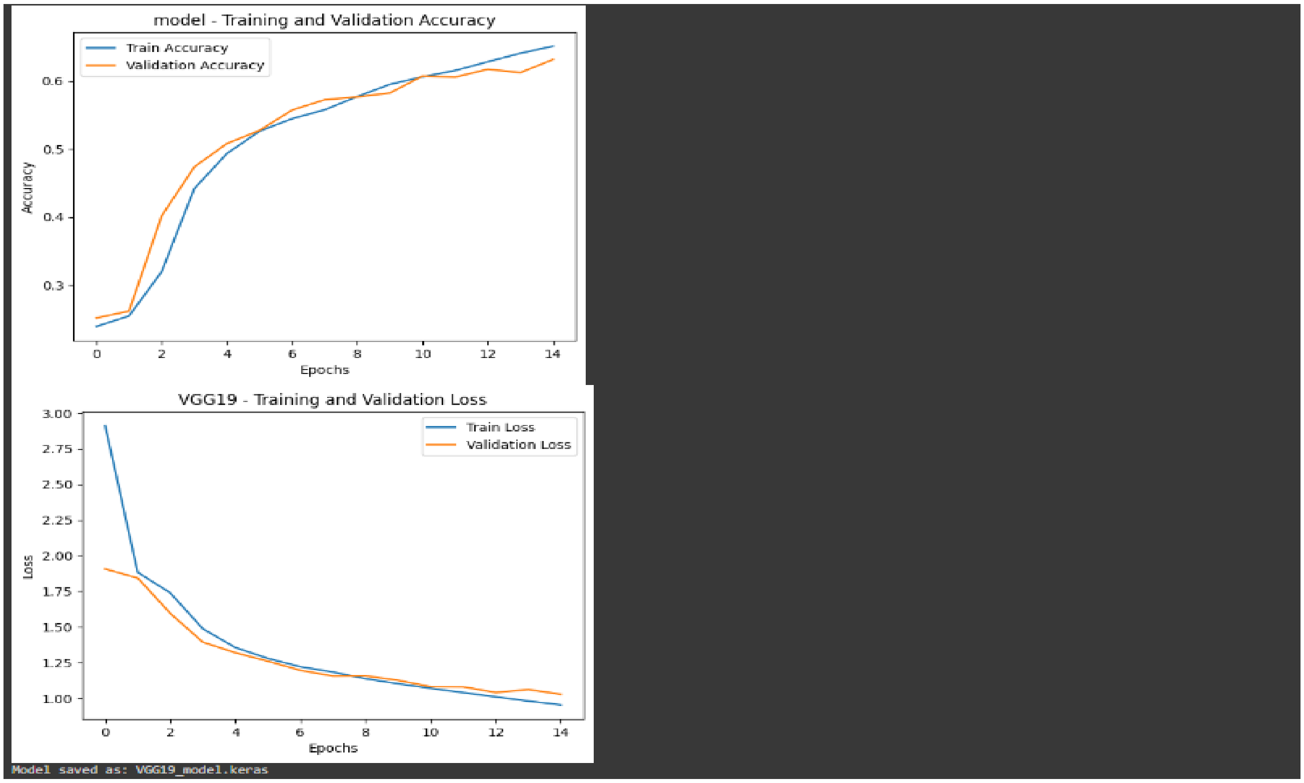
b) VGG19

- **Creating and training the model:**

```python
from keras.optimizers import Adam
from keras.applications import VGG16, VGG19, ResNet50, ResNet152, MobileNet

train_gen, val_gen, test_gen = get_generators((224, 224))

input_shape = (224, 224, 1)

# VGG19
VGG19_model = create_model(VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3)), 7, input_shape, l2_reg=0.01, dropout_rate=0.5)
VGG19_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
history_VGG19 = train_and_evaluate(VGG19_model, train_gen, val_gen, test_gen, "VGG19")
```

```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 4s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 1)]     0

 conv2d (Conv2D)             (None, 224, 224, 3)       30

 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 1024)              525312

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 7)                 7175

=================================================================
Total params: 20556901 (78.42 MB)
Trainable params: 20556901 (78.42 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
_________________________________________________________________
Epoch 1/15
1436/1436 [==============================] - 430s 284ms/step - loss: 2.9121 - accuracy: 0.2388 - val_loss: 1.9074 - val_accuracy: 0.2513 - lr: 1.0000e-04
Epoch 2/15
1436/1436 [==============================] - 399s 278ms/step - loss: 1.8835 - accuracy: 0.2543 - val_loss: 1.8432 - val_accuracy: 0.2615 - lr: 1.0000e-04
Epoch 3/15
1436/1436 [==============================] - 399s 278ms/step - loss: 1.7385 - accuracy: 0.3193 - val_loss: 1.5953 - val_accuracy: 0.4013 - lr: 1.0000e-04
Epoch 4/15
1436/1436 [==============================] - 400s 278ms/step - loss: 1.4872 - accuracy: 0.4417 - val_loss: 1.3933 - val_accuracy: 0.4736 - lr: 1.0000e-04
Epoch 5/15
1436/1436 [==============================] - 400s 278ms/step - loss: 1.3557 - accuracy: 0.4933 - val_loss: 1.3196 - val_accuracy: 0.5079 - lr: 1.0000e-04
Epoch 6/15
1436/1436 [==============================] - 400s 279ms/step - loss: 1.2799 - accuracy: 0.5261 - val_loss: 1.2600 - val_accuracy: 0.5273 - lr: 1.0000e-04
Epoch 7/15
1436/1436 [==============================] - 400s 278ms/step - loss: 1.2210 - accuracy: 0.5445 - val_loss: 1.1959 - val_accuracy: 0.5572 - lr: 1.0000e-04
Epoch 8/15
1436/1436 [==============================] - 400s 278ms/step - loss: 1.1839 - accuracy: 0.5576 - val_loss: 1.1562 - val_accuracy: 0.5724 - lr: 1.0000e-04
Epoch 9/15
1436/1436 [==============================] - 409s 284ms/step - loss: 1.1380 - accuracy: 0.5771 - val_loss: 1.1577 - val_accuracy: 0.5766 - lr: 1.0000e-04
Epoch 10/15
1436/1436 [==============================] - 401s 279ms/step - loss: 1.1023 - accuracy: 0.5950 - val_loss: 1.1267 - val_accuracy: 0.5823 - lr: 1.0000e-04
Epoch 11/15
1436/1436 [==============================] - 400s 278ms/step - loss: 1.0701 - accuracy: 0.6061 - val_loss: 1.0816 - val_accuracy: 0.6072 - lr: 1.0000e-04
Epoch 12/15
1436/1436 [==============================] - 400s 279ms/step - loss: 1.0402 - accuracy: 0.6153 - val_loss: 1.0805 - val_accuracy: 0.6056 - lr: 1.0000e-04
Epoch 13/15
1436/1436 [==============================] - 401s 279ms/step - loss: 1.0107 - accuracy: 0.6284 - val_loss: 1.0409 - val_accuracy: 0.6171 - lr: 1.0000e-04
Epoch 14/15
1436/1436 [==============================] - 400s 278ms/step - loss: 0.9813 - accuracy: 0.6410 - val_loss: 1.0617 - val_accuracy: 0.6123 - lr: 1.0000e-04
Epoch 15/15
1436/1436 [==============================] - 401s 279ms/step - loss: 0.9547 - accuracy: 0.6513 - val_loss: 1.0284 - val_accuracy: 0.6318 - lr: 1.0000e-04
```



- **Testing the model with random images in the test data:**

```python
from keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# Load the saved MobileNet model
VGG19_model = load_model("VGG19_model.keras")

# Emotion classes for the dataset
Emotion_Classes = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Assuming test_gen and model are already defined
batch_size = test_gen.batch_size

# Setting the random seed
np.random.seed()

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_gen) - 1)

# Selecting random image indices from the batch
Random_Img_Index = np.random.randint(0, batch_size, 10)

# Setting up the plot
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    # Fetching the random image and its label
    Random_Img = test_gen[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_gen[Random_batch][1][Random_Img_Index[i]], axis=0)

    # Making a prediction using the model
    Model_Prediction = np.argmax(VGG19_model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0), axis=1)[0]
    print(VGG19_model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0))

    # Displaying the image
    ax.imshow(Random_Img.squeeze(), cmap='gray')  # Assuming the images are grayscale
    # Setting the title with true and predicted labels, colored based on correctness
    color = "green" if Emotion_Classes[Random_Img_Label] == Emotion_Classes[Model_Prediction] else "red"
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classes[Model_Prediction]}", color=color)

plt.tight_layout()
plt.show()
```

- **Evaluate the accuracy of the model using test data:**

```
# Evaluate on the test set and calculate additional metrics
test_loss, test_acc = VGG19_model.evaluate(test_gen)
# Get predictions and ground truth labels from the test set
predictions = VGG19_model.predict(test_gen)
predicted_classes = np.argmax(predictions, axis=1)  # Get class with highest probability
true_classes = test_gen.classes

# Calculate precision, recall, and F1-score
precision = precision_score(true_classes, predicted_classes, average='macro')
recall = recall_score(true_classes, predicted_classes, average='macro')
f1 = f1_score(true_classes, predicted_classes, average='macro')

# Print precision, recall, and F1 score
print("Accuracy: ", test_acc)
print("Loss: ", test_loss)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
449/449 [==============================] - 39s 87ms/step - loss: 1.0183 - accuracy: 0.6353
449/449 [==============================] - 35s 77ms/step
Accuracy:  0.6352744698524475
Loss:  1.0183348655700684
Precision: 0.14777817281879754
Recall: 0.14518558605559315
F1 Score: 0.14069999449073273
```

- **Link to the Model:**

  https://colab.research.google.com/drive/12MVgcFytLc0JuPJtM5x6QLOY-uomsKg4?usp=sharing

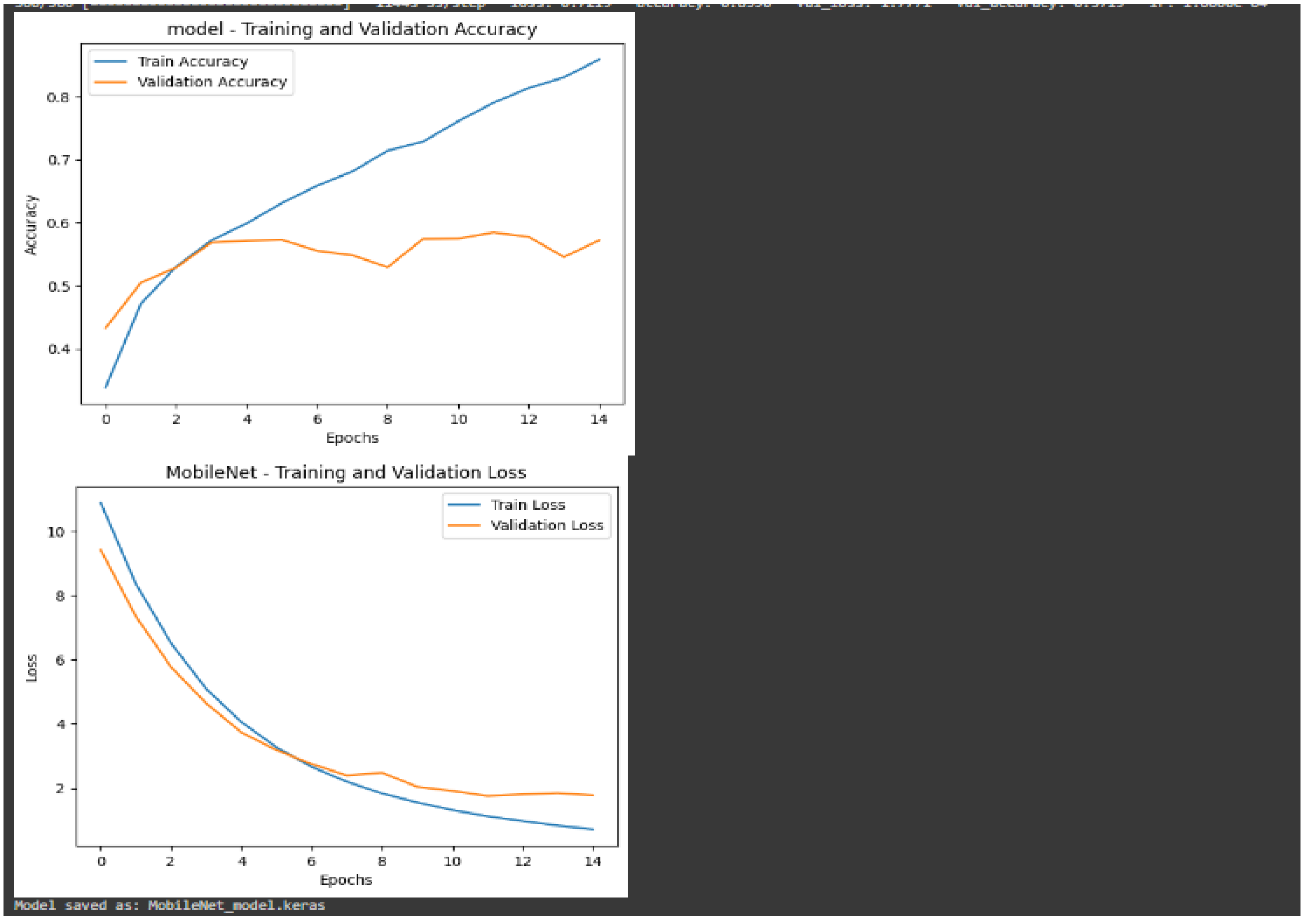
c) MobileNet model

- **Creating and training the model:**

```python
from keras.optimizers import Adam
from keras.applications import VGG16, VGG19, ResNet50, ResNet152, MobileNet

# Define generators once
train_gen, val_gen, test_gen = get_generators((224, 224))

input_shape = (224, 224, 1)

# MobileNet
mobilenet_model = create_model(MobileNet(weights='imagenet', include_top=False, input_shape=(224, 224, 3)), 7, input_shape, l2_reg=0.01, dropout_rate=0.5)
mobilenet_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
mobilenet_history = train_and_evaluate(mobilenet_model, train_gen, val_gen, test_gen, "MobileNet")
```

```
Found 5746 images belonging to 7 classes.
Found 1432 images belonging to 7 classes.
Found 28709 images belonging to 7 classes.
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 1)]     0

 conv2d (Conv2D)             (None, 224, 224, 3)       30

 mobilenet_1.00_224 (Functi  (None, 7, 7, 1024)        3228864
 onal)

 global_average_pooling2d (  (None, 1024)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 1024)              1049600

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 7)                 7175

=================================================================
Total params: 4285669 (16.35 MB)
Trainable params: 4263781 (16.27 MB)
Non-trainable params: 21888 (85.50 KB)
```

```
Epoch 1/15
360/360 [==============================] - 1168s 3s/step - loss: 10.8724 - accuracy: 0.3390 - val_loss: 9.4089 - val_accuracy: 0.4330 - lr: 1.0000e-04
Epoch 2/15
360/360 [==============================] - 1167s 3s/step - loss: 8.3459 - accuracy: 0.4716 - val_loss: 7.3356 - val_accuracy: 0.5049 - lr: 1.0000e-04
Epoch 3/15
360/360 [==============================] - 1192s 3s/step - loss: 6.5134 - accuracy: 0.5299 - val_loss: 5.7683 - val_accuracy: 0.5286 - lr: 1.0000e-04
Epoch 4/15
360/360 [==============================] - 1161s 3s/step - loss: 5.0913 - accuracy: 0.5719 - val_loss: 4.6424 - val_accuracy: 0.5684 - lr: 1.0000e-04
Epoch 5/15
360/360 [==============================] - 1165s 3s/step - loss: 4.0567 - accuracy: 0.5989 - val_loss: 3.7332 - val_accuracy: 0.5712 - lr: 1.0000e-04
Epoch 6/15
360/360 [==============================] - 1140s 3s/step - loss: 3.2690 - accuracy: 0.6312 - val_loss: 3.1821 - val_accuracy: 0.5726 - lr: 1.0000e-04
Epoch 7/15
360/360 [==============================] - 1137s 3s/step - loss: 2.6608 - accuracy: 0.6587 - val_loss: 2.7499 - val_accuracy: 0.5552 - lr: 1.0000e-04
Epoch 8/15
360/360 [==============================] - 1131s 3s/step - loss: 2.2022 - accuracy: 0.6812 - val_loss: 2.3966 - val_accuracy: 0.5482 - lr: 1.0000e-04
Epoch 9/15
360/360 [==============================] - 1168s 3s/step - loss: 1.8358 - accuracy: 0.7142 - val_loss: 2.4819 - val_accuracy: 0.5293 - lr: 1.0000e-04
Epoch 10/15
360/360 [==============================] - 1198s 3s/step - loss: 1.5581 - accuracy: 0.7285 - val_loss: 2.0433 - val_accuracy: 0.5740 - lr: 1.0000e-04
Epoch 11/15
360/360 [==============================] - 1142s 3s/step - loss: 1.3169 - accuracy: 0.7611 - val_loss: 1.9144 - val_accuracy: 0.5747 - lr: 1.0000e-04
Epoch 12/15
360/360 [==============================] - 1148s 3s/step - loss: 1.1272 - accuracy: 0.7903 - val_loss: 1.7547 - val_accuracy: 0.5845 - lr: 1.0000e-04
Epoch 13/15
360/360 [==============================] - 1143s 3s/step - loss: 0.9752 - accuracy: 0.8136 - val_loss: 1.8072 - val_accuracy: 0.5775 - lr: 1.0000e-04
Epoch 14/15
360/360 [==============================] - 1148s 3s/step - loss: 0.8346 - accuracy: 0.8307 - val_loss: 1.8441 - val_accuracy: 0.5454 - lr: 1.0000e-04
Epoch 15/15
360/360 [==============================] - 1144s 3s/step - loss: 0.7219 - accuracy: 0.8590 - val_loss: 1.7771 - val_accuracy: 0.5719 - lr: 1.0000e-04
```



- **Testing the model with random images in the test data:**

```
from keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# Load the saved MobileNet model
mobilenet_model = load_model("MobileNet_model.keras")

# Emotion classes for the dataset
Emotion_Classes = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Assuming test_gen and model are already defined
batch_size = test_gen.batch_size

# Setting the random seed
np.random.seed()

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_gen) - 1)

# Selecting random image indices from the batch
Random_Img_Index = np.random.randint(0, batch_size, 10)

# Setting up the plot
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    # Fetching the random image and its label
    Random_Img = test_gen[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_gen[Random_batch][1][Random_Img_Index[i]], axis=0)

    # Making a prediction using the model
    Model_Prediction = np.argmax(mobilenet_model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0), axis=1)[0]

    # Displaying the image
    ax.imshow(Random_Img.squeeze(), cmap='gray')  # Assuming the images are grayscale
    # Setting the title with true and predicted labels, colored based on correctness
    color = "green" if Emotion_Classes[Random_Img_Label] == Emotion_Classes[Model_Prediction] else "red"
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classes[Model_Prediction]}", color=color)

plt.tight_layout()
plt.show()
```

- **Evaluate the accuracy of the model using test data:**

```
mobilenet_model = load_model("MobileNet_model.keras")

# Evaluate on the test set and calculate additional metrics
test_loss, test_acc = mobilenet_model.evaluate(test_gen)
# Get predictions and ground truth labels from the test set
predictions = mobilenet_model.predict(test_gen)
predicted_classes = np.argmax(predictions, axis=1)  # Get class with highest probability
true_classes = test_gen.classes

# Calculate precision, recall, and F1-score
precision = precision_score(true_classes, predicted_classes, average='macro')
recall = recall_score(true_classes, predicted_classes, average='macro')
f1 = f1_score(true_classes, predicted_classes, average='macro')

# Print precision, recall, and F1 score
print("Accuracy: ", test_acc)
print("Loss: ", test_loss)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
1795/1795 [==============================] - 1123s 625ms/step - loss: 1.7582 - accuracy: 0.5733
1795/1795 [==============================] - 1123s 625ms/step
Accuracy:  0.5732697248458862
Loss:  1.7581883668899536
Precision: 0.1454864380702481
Recall: 0.14529612125922862
F1 Score: 0.14317380650314626
```

- **Link to the Model:**

https://colab.research.google.com/drive/18qkn0iEMf9FGS1ogo7DQMRgDx4SbeJWM?usp=sharing

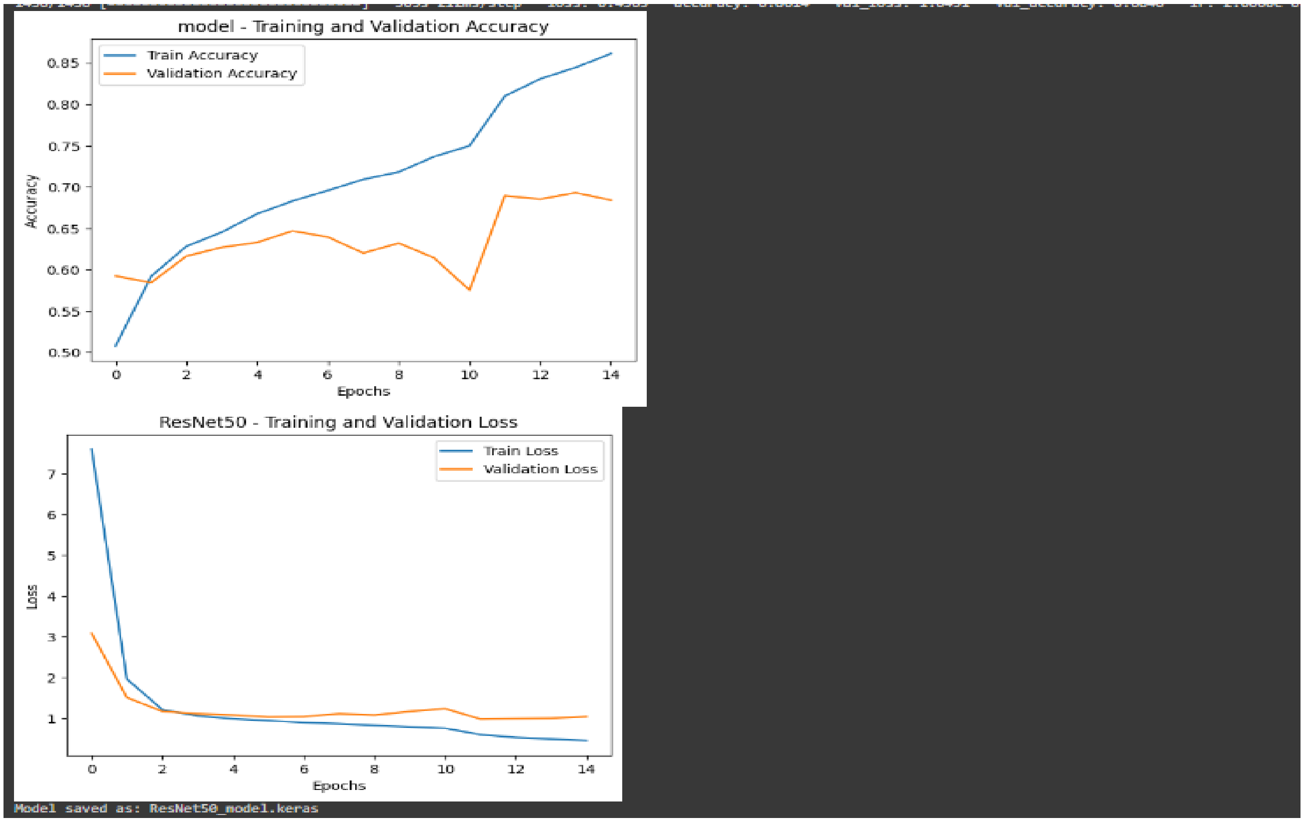
d) ResNet50 Model

- ### Creating and training the model:

```
from keras.optimizers import Adam
from keras.applications import VGG16, VGG19, ResNet50, ResNet152, MobileNet

train_gen, val_gen, test_gen = get_generators((224, 224))

input_shape = (224, 224, 1)

# ResNet50
resnet50_model = create_model(ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3)), 7, input_shape, l2_reg=0.01, dropout_rate=0.5)
resnet50_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
resnet_history = train_and_evaluate(resnet50_model, train_gen, val_gen, test_gen, "ResNet50")
```

```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 0s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 1)]     0

 conv2d (Conv2D)             (None, 224, 224, 3)       30

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 1024)              2098176

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 7)                 7175

=================================================================
Total params: 25693093 (98.01 MB)
Trainable params: 25639973 (97.81 MB)
Non-trainable params: 53120 (207.50 KB)
```

```
_________________________________________________________________
Epoch 1/15
1436/1436 [==============================] - 351s 215ms/step - loss: 7.6025 - accuracy: 0.5079 - val_loss: 3.0870 - val_accuracy: 0.5924 - lr: 1.0000e-04
Epoch 2/15
1436/1436 [==============================] - 318s 221ms/step - loss: 1.9630 - accuracy: 0.5920 - val_loss: 1.5102 - val_accuracy: 0.5844 - lr: 1.0000e-04
Epoch 3/15
1436/1436 [==============================] - 306s 213ms/step - loss: 1.2130 - accuracy: 0.6283 - val_loss: 1.1724 - val_accuracy: 0.6164 - lr: 1.0000e-04
Epoch 4/15
1436/1436 [==============================] - 306s 213ms/step - loss: 1.0626 - accuracy: 0.6454 - val_loss: 1.1149 - val_accuracy: 0.6271 - lr: 1.0000e-04
Epoch 5/15
1436/1436 [==============================] - 306s 213ms/step - loss: 0.9905 - accuracy: 0.6675 - val_loss: 1.0809 - val_accuracy: 0.6330 - lr: 1.0000e-04
Epoch 6/15
1436/1436 [==============================] - 305s 213ms/step - loss: 0.9446 - accuracy: 0.6830 - val_loss: 1.0390 - val_accuracy: 0.6469 - lr: 1.0000e-04
Epoch 7/15
1436/1436 [==============================] - 305s 212ms/step - loss: 0.8970 - accuracy: 0.6958 - val_loss: 1.0446 - val_accuracy: 0.6394 - lr: 1.0000e-04
Epoch 8/15
1436/1436 [==============================] - 305s 212ms/step - loss: 0.8674 - accuracy: 0.7092 - val_loss: 1.1141 - val_accuracy: 0.6201 - lr: 1.0000e-04
Epoch 9/15
1436/1436 [==============================] - 305s 213ms/step - loss: 0.8285 - accuracy: 0.7182 - val_loss: 1.0814 - val_accuracy: 0.6321 - lr: 1.0000e-04
Epoch 10/15
1436/1436 [==============================] - 305s 212ms/step - loss: 0.7938 - accuracy: 0.7367 - val_loss: 1.1707 - val_accuracy: 0.6144 - lr: 1.0000e-04
Epoch 11/15
1436/1436 [==============================] - 305s 212ms/step - loss: 0.7590 - accuracy: 0.7498 - val_loss: 1.2392 - val_accuracy: 0.5753 - lr: 1.0000e-04
Epoch 12/15
1436/1436 [==============================] - 316s 220ms/step - loss: 0.6018 - accuracy: 0.8099 - val_loss: 0.9841 - val_accuracy: 0.6894 - lr: 2.0000e-05
Epoch 13/15
1436/1436 [==============================] - 316s 220ms/step - loss: 0.5321 - accuracy: 0.8307 - val_loss: 0.9944 - val_accuracy: 0.6852 - lr: 2.0000e-05
Epoch 14/15
1436/1436 [==============================] - 316s 220ms/step - loss: 0.4961 - accuracy: 0.8444 - val_loss: 1.0035 - val_accuracy: 0.6931 - lr: 2.0000e-05
Epoch 15/15
1436/1436 [==============================] - 305s 212ms/step - loss: 0.4565 - accuracy: 0.8614 - val_loss: 1.0451 - val_accuracy: 0.6840 - lr: 2.0000e-05
```



- **Testing the model with random images in the test data:**

```python
from keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# Load the saved MobileNet model
resnet50_model = load_model("ResNet50_model.keras")

# Emotion classes for the dataset
Emotion_Classes = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Assuming test_gen and model are already defined
batch_size = test_gen.batch_size

# Setting the random seed
np.random.seed()

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_gen) - 1)

# Selecting random image indices from the batch
Random_Img_Index = np.random.randint(0, batch_size, 10)

# Setting up the plot
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    # Fetching the random image and its label
    Random_Img = test_gen[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_gen[Random_batch][1][Random_Img_Index[i]], axis=0)

    # Making a prediction using the model
    Model_Prediction = np.argmax(resnet50_model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0), axis=1)[0]
    print(resnet50_model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0))

    # Displaying the image
    ax.imshow(Random_Img.squeeze(), cmap='gray')  # Assuming the images are grayscale
    # Setting the title with true and predicted labels, colored based on correctness
    color = "green" if Emotion_Classes[Random_Img_Label] == Emotion_Classes[Model_Prediction] else "red"
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classes[Model_Prediction]}", color=color)

plt.tight_layout()
plt.show()
```

- **Evaluate the accuracy of the model using test data:**

```python
# Evaluate on the test set and calculate additional metrics
test_loss, test_acc = resnet50_model.evaluate(test_gen)
# Get predictions and ground truth labels from the test set
predictions = resnet50_model.predict(test_gen)
predicted_classes = np.argmax(predictions, axis=1)  # Get class with highest probability
true_classes = test_gen.classes

# Calculate precision, recall, and F1-score
precision = precision_score(true_classes, predicted_classes, average='macro')
recall = recall_score(true_classes, predicted_classes, average='macro')
f1 = f1_score(true_classes, predicted_classes, average='macro')

# Print precision, recall, and F1 score
print("Accuracy: ", test_acc)
print("Loss: ", test_loss)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
449/449 [==============================] - 28s 60ms/step - loss: 1.0944 - accuracy: 0.6863
449/449 [==============================] - 25s 53ms/step
Accuracy:  0.6862635612487793
Loss:  1.0943892002105713
Precision: 0.14794058892703976
Recall: 0.1484758226988762
F1 Score: 0.1478689553778462
```

- **Link to the Model:**

https://colab.research.google.com/drive/1vugzTR14pxoQToks636U-XEOeBAMrY0P?usp=sharing

7. Conclusion:

a) ResNet

- Accuracy: 0.68
- Loss: 1.094

ResNet has the highest accuracy among all the models tested, indicating that it is the most effective at correctly classifying the test data. However, its loss is relatively high compared to VGG16 and VGG19, suggesting that while it often predicts correctly, the confidence of its predictions (as measured by the loss function) may not be as high.

b) VGG16

- Accuracy: 0.645
- Loss: 1.0299

VGG16 shows slightly lower accuracy than ResNet but has a lower loss. This indicates that VGG16's predictions, while slightly less accurate, are more confident and closer to the actual values compared to ResNet. It strikes a good balance between accuracy and loss, making it a reliable model for this dataset.

c) VGG19

- Accuracy: 0.635
- Loss: 1.0183

VGG19 has a marginally lower accuracy compared to VGG16 but has the lowest loss among all models. This suggests that VGG19's predictions are the most confident, even though it is slightly less accurate than ResNet and VGG16. The lower loss indicates that the model's predictions are closer to the actual values, even if the number of correct classifications is slightly lower.

d) MobileNet

- Accuracy: 0.573
- Loss: 1.758

MobileNet has the lowest accuracy and the highest loss among all the models. This indicates that it is the least effective at correctly classifying the test data and its predictions are the least confident. MobileNet's performance is significantly worse than the other models, making it the least suitable for this dataset.

Summary

- **Best Overall Model:** ResNet, due to its highest accuracy, making it the most effective for classification tasks on this dataset.

- **Best Model in Terms of Loss:** VGG19, which has the lowest loss, indicating the most confident predictions.

- **Balanced Performance:** VGG16, which offers a good trade-off between accuracy and loss.

- **Least Effective Model:** MobileNet, due to its low accuracy and high loss, suggesting it is not well-suited for this particular dataset.

In conclusion, ResNet is the preferred choice if accuracy is the primary goal. If minimizing prediction error and confidence is more critical, VGG19 might be considered. VGG16 offers a balanced approach, while MobileNet appears to be the least effective for this specific task.

8. Dataset Link:

https://www.kaggle.com/datasets/msambare/fer2013/data?select=test

https://www.kaggle.com/datasets/apollo2506/facial-recognition-dataset