📋 TABLE OF CONTENTS

- 1. Emotion Classifier from scratch using Keras: Intro
- 2. Dataset
- 3. Load Kaggle Dataset in Google Colab
- 4. Installing the required libraries
- 5. Importing packages
- 6. Create Project Directory Structure to save models
- 7. Data Cleaning: Removing corrupted images
- 8. Data Analysis
- 9. Plotting images from dataset
  - 1. Plotting one image from each emotion folder
  - 2. Plotting random images from given emotion folder
- 10. Checking image shapes and channels
- 11. Model 1 : Custom CNN from Scratch
  - 1. Data Loading: Load Images using `Keras ImageDataGenerator`
  - 2. Getting Class Labels
  - 3. Building Model
  - 4. Understanding Callbacks
    - 1. `ModelCheckpoint` Callback
    - 2. `EarlyStopping` Callback
    - 3. `ReduceLROnPlateau` Callback
    - 4. `CSVLogger` Callback
  - 5. Training Model
  - 6. Plotting Performance Metrics
  - 7. Model Evaluation
  - 8. Confusion Matrix
  - 9. Classification Report
  - 10. Making Predictions
  - 11. Conclusion
- 12. Model 2 : Custom CNN with Image Augmentation
  - 1. Data Loading: Load Images using `Keras ImageDataGenerator` with Data Augmentation
  - 2. Getting Class Labels
  - 3. Building Model: Same CNN Model as above
  - 4. Callbacks
  - 5. Training Model
  - 6. Plotting Performance Metrics
  - 7. Model Evaluation
  - 8. Confusion Matrix
  - 9. Classification Report
  - 10. Making Predictions
  - 11. Conclusion
- 13. Transfer Learning and Fine-tuning
  - 1. Background
  - 2. Transfer Learning

    - 3. Transfer Learning in Action
- 14. VGG16 Model
    - 1. References
    - 2. Intro
    - 3. VGG16 Architecture
- 15. Model 3 : Transfer Learning VGG16
    - 1. Data Loading: Load Images using `Keras ImageDataGenerator` with Data Augmentation
    - 2. Introduction of Class Weights to take care of Dataset Imbalance
    - 3. Building Model
    - 4. Callbacks
    - 5. Training Model
    - 6. Plotting Performance Metrics
    - 7. Model Evaluation
    - 8. Confusion Matrix
    - 9. Classification Report
    - 10. Making Predictions
    - 11. Conclusion
- 16. ResNet50 Model
    - 1. References
    - 2. ResNet Model: Intro
    - 3. Residual Learning
    - 4. Comparison of Plain and Residual Networks
    - 5. Deeper Bottleneck Architectures
    - 6. ResNet Architectures for ImageNet
    - 7. ResNet50 Architecture
- 17. Model 4 : Transfer Learning ResNet50
    - 1. Data Loading: Load Images using `Keras ImageDataGenerator` with Data Augmentation
    - 2. Introduction of Class Weights to take care of Dataset Imbalance
    - 3. Building Model
    - 4. Callbacks
    - 5. Training Model
    - 6. Plotting Performance Metrics
    - 7. Model Evaluation
    - 8. Confusion Matrix
    - 9. Classification Report
    - 10. ROC Plots for all classes
    - 11. Making Predictions
    - 12. Conclusion

---

1. Emotion Classifier from scratch using Keras: Intro

We are going to train an Emotion Classifier from scratch using Keras on the Kaggle FER-2013 (Facial Emotion Recognition) Dataset.

2. Dataset

Kaggle FER-2013 dataset consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image.

The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the test set consists of 3,589 examples.

The dataset has got lots of challenges that needs to be addressed:

- 1st challenge: Learning from a big image is easier compared to learning from a small image (48x48 pixels)
- 2nd challenge: We have many classes
- 3rd challenge: We have less no of images per class
- 4th challenge: Dataset is highly imbalanced (Train set has got 400+ images for 'disgust' class while 7200+ for 'happy' class)

3. Load Kaggle Dataset in Google Colab

In [ ]:
```
# Install Kaggle
!pip3 install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.1
4)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (fro
m kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-pack
ages (from kaggle) (2024.6.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-package
s (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kag
gle) (4.66.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages
(from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from
kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from k
aggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages
(from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-pac
kages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
t-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
(from requests->kaggle) (3.7)
```

In [ ]:
```
# Upload Kaggle API key (kaggle.json file downloaded from your Kaggle account)
from google.colab import files
files.upload()
```

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
```

Out[ ]:
```
{'kaggle.json': b'{"username":"ancilcleetus","key":"21c225043e2abdc810ba33e0243b41e6"}'}
```

In [ ]:
```
# Move kaggle.json file to .kaggle folder in home directory
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# Change the permission of the file
!chmod 600 ~/.kaggle/kaggle.json

# Check the permission of the file
!ls -l ~/.kaggle/kaggle.json
```

```
-rw------- 1 root root 68 Jun 28 02:53 /root/.kaggle/kaggle.json
```

In [ ]:
```
# Get Kaggle FER-2013 Dataset API Command & Download dataset
!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
 96% 58.0M/60.3M [00:03<00:00, 24.7MB/s]
100% 60.3M/60.3M [00:03<00:00, 17.2MB/s]
```

In [ ]:
```
# Extract dataset zip file
from zipfile import ZipFile
file_name = "fer2013.zip"

with ZipFile(file_name, 'r') as zip_file:
  zip_file.extractall()
  print("Done")
```

```
Done
```

4. Installing the required libraries

In [ ]:
```
!pip3 install tensorflow keras numpy matplotlib opencv-python
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.
15.0)
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.
7.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages
(4.8.0.76)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages
(from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packa
ges (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-pa
ckages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/pyt
hon3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-pac
```

```
kages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (f
rom tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packag
es (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packag
es (from tensorflow) (0.2.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packa
ges (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (fro
m tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.
21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.
20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fr
om tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (f
rom tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packag
es (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dis
t-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/py
thon3.10/dist-packages (from tensorflow) (0.37.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist
-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/pyth
on3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages
(from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib) (4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
(from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib) (2.8.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-pack
ages (from astunparse>=1.6.0->tensorflow) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-p
ackages (from tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.1
0/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-package
s (from tensorboard<2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-pac
kages (from tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/p
ython3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-package
s (from tensorboard<2.16,>=2.15->tensorflow) (3.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-
packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-p
ackages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages
(from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (4.9)
```

```
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2024.6.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
```

---

5. Importing packages

In [ ]:

```python
import os
import random

import numpy as np
import pandas as pd

from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import imghdr

import tensorflow as tf
from tensorflow.keras import layers, models, regularizers, optimizers
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Dropout, F
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_ar
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16, ResNet50V2
from keras.utils import plot_model
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from sklearn.utils.class_weight import compute_class_weight
```

---

6. Create Project Directory Structure to save models

In [ ]:

```python
# Define your project name
project_name = 'FER_2013_Emotion_Classifier'

# List your models
model_names = [
    'Custom_CNN_From_Scratch',
```

```
    'Custom_CNN_With_Augmentation',
    'VGG16_Transfer_Learning',
    'ResNet50_Transfer_Learning'
]

# Base directory (in this case, your Google Colab workspace)
base_dir = '/content/'

# Create the project directory
project_dir = os.path.join(base_dir, project_name)
os.makedirs(project_dir, exist_ok=True)

# Create a subdirectory for each model
for each_model in model_names:
    model_dir = os.path.join(project_dir, each_model)
    os.makedirs(model_dir, exist_ok=True)
    # Example subdirectories for model-related files
    # os.makedirs(os.path.join(model_dir, 'checkpoints'), exist_ok=True)
    # os.makedirs(os.path.join(model_dir, 'logs'), exist_ok=True)
    # os.makedirs(os.path.join(model_dir, 'saved_models'), exist_ok=True)

print(f'Project directory structure created at: {project_dir}')
```

```
Project directory structure created at: /content/FER_2013_Emotion_Classifier
```

---

7. Data Cleaning: Removing corrupted images

We will not be performing this step.

In [ ]:

```
# Define the list of acceptable image extensions
image_exts = ['jpeg', 'jpg', 'png']

# Path to the directory containing image classes and possibly other nested subdirectorie
data_dir = '/content/train'

# Walk through all directories and files in the dataset
for root, dirs, files in os.walk(data_dir):
    for file in files:
        # Construct the path to the current file
        file_path = os.path.join(root, file)

        try:
            # Check the file type of the current file
            file_type = imghdr.what(file_path)

            # If the file extension is not in the allowed list, remove it
            if file_type not in image_exts:
                print(f'Image not in ext list {file_path}')
                os.remove(file_path)
            else:
                # Proceed to process the image if needed, for example, reading it with O
                img = cv2.imread(file_path)

        except Exception as e:
            # Print out the issue and the path of the problematic file
            print(f'Issue with file {file_path}. Error: {e}')
            # Optionally, remove files that cause exceptions
            os.remove(file_path)
```

---

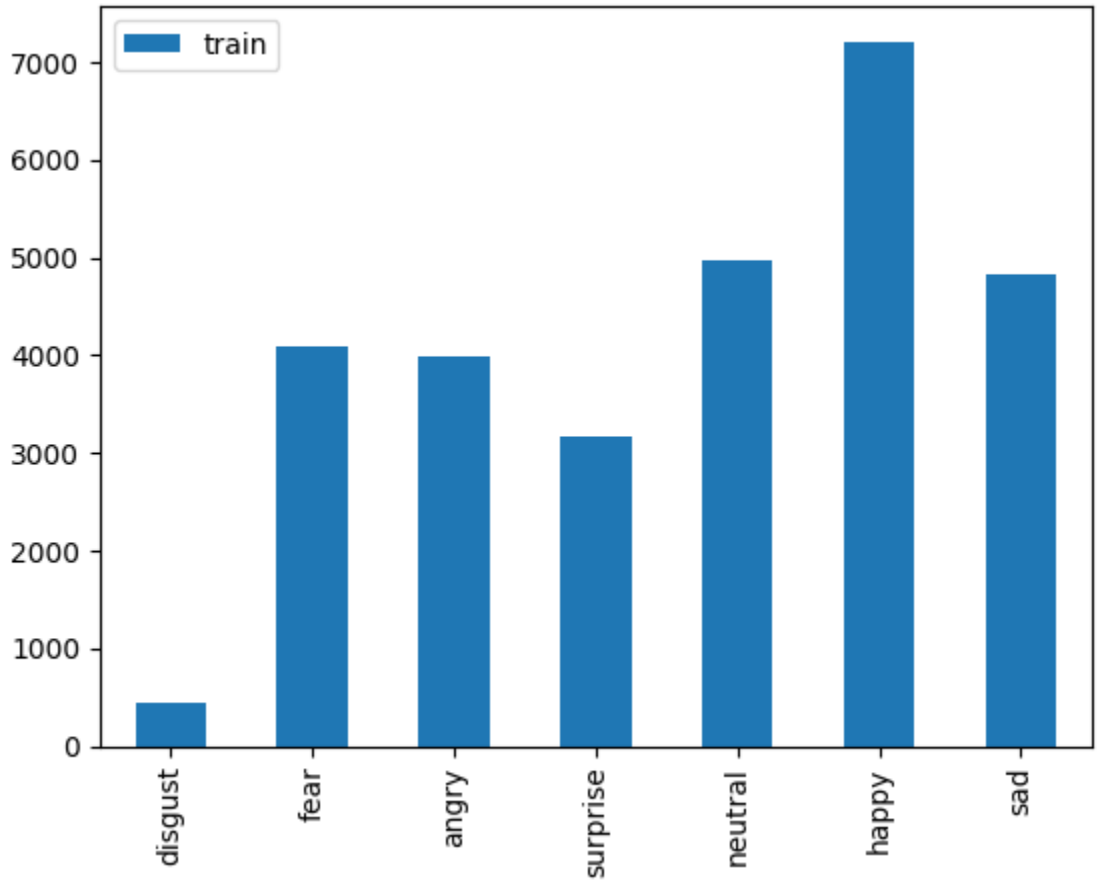
8. Data Analysis

In [ ]:
```python
# Define a function to count the number of files (assumed to be images for this context)
# The function returns a DataFrame with these counts, indexed by a specified set name (e
def count_files_in_subdirs(directory, set_name):
    # Initialize an empty dictionary to hold the count of files for each subdirectory.
    counts = {}

    # Iterate over each item in the given directory
    for item in os.listdir(directory):
        # Construct the full path to the item.
        item_path = os.path.join(directory, item)

        # Check if the item is a directory
        if os.path.isdir(item_path):
            # Count the number of files in the subdirectory and add it to the dictionary
            counts[item] = len(os.listdir(item_path))

    # Convert the counts dictionary to a DataFrame for easy viewing and analysis
    # The index of the DataFrame is set to the provided set name
    df = pd.DataFrame(counts, index=[set_name])
    return df

# Paths to the training and testing directories
train_dir = '/content/train'
test_dir = '/content/test'

# Count the files in the subdirectories of the training directory and print the result
train_count = count_files_in_subdirs(train_dir, 'train')
print(train_count)

# Count the files in the subdirectories of the testing directory and print the result
test_count = count_files_in_subdirs(test_dir, 'test')
print(test_count)
```

```
       disgust  fear  angry  surprise  neutral  happy   sad
train      436  4097   3995      3171     4965   7215  4830
      disgust  fear  angry  surprise  neutral  happy   sad
test      111  1024    958       831     1233   1774  1247
```

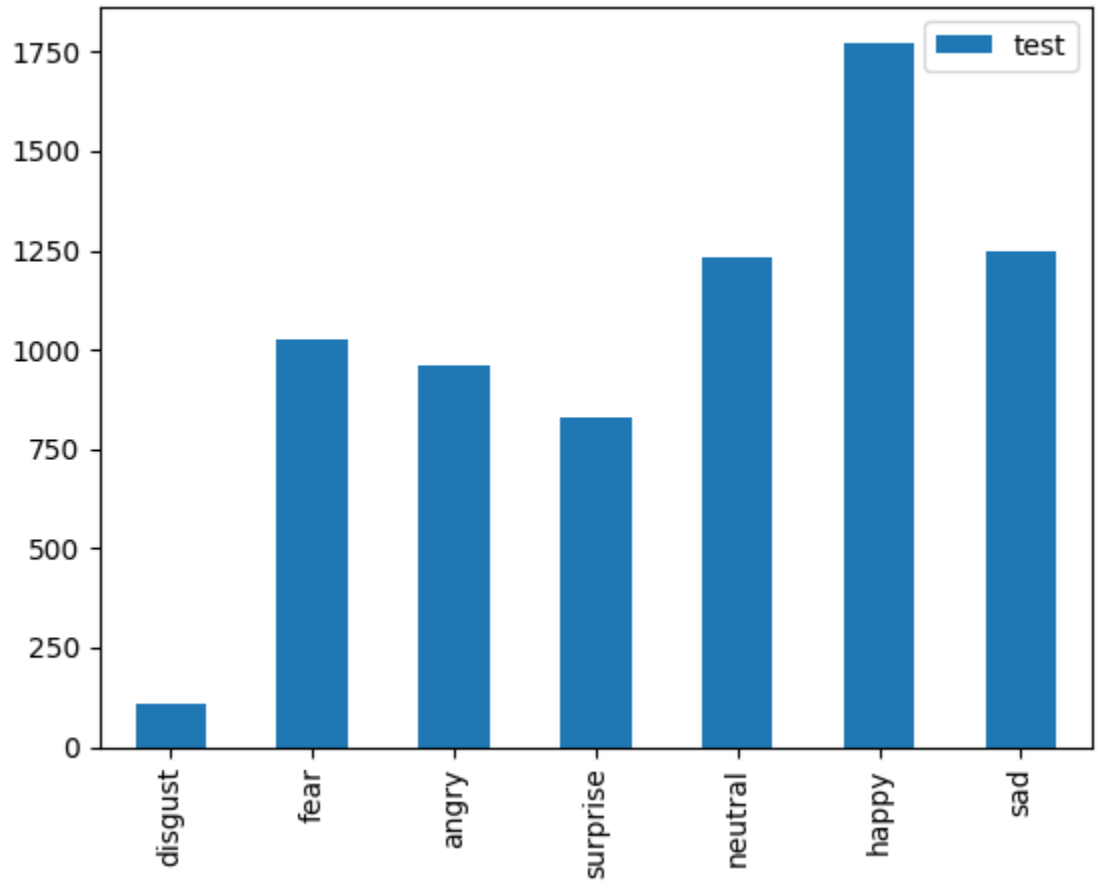
In [ ]:
```python
train_count.transpose().plot(kind='bar')
```

Out[ ]:
```
<Axes: >
```



In [ ]:
```
test_count.transpose().plot(kind='bar')
```

Out[ ]:
```
<Axes: >
```



In the Training dataset, `disgust` class has got very few images (400+) while `happy` class has got very high number of images (7200+). Hence, model will learn more things about `happy` people while learning

very less about `disgust` people. This is not what we want. This problem is called Class Imbalance.

In this scenario, we cannot rely solely on Accuracy metric since it tells only about the overall accuracy and does not account for the distribution of classes. In an imbalanced dataset, a model might achieve high accuracy by correctly predicting the majority class while poorly predicting the minority classes.

To evaluate the performance of our model more comprehensively, we can use Precision, Recall and F1-Score metrics.

- Precision (Positive Predictive Value):
    - The proportion of true positive predictions among all positive predictions. Useful when the cost of false positives is high.
    - Precision is the ratio of true positive predictions to the total number of positive predictions (true positives + false positives).
    - Precision = TP/(TP + FP) where
        - TP = True Positives (correctly predicted positive instances)
        - FP = False Positives (incorrectly predicted positive instances)
- Recall (Sensitivity or True Positive Rate):
    - The proportion of true positive predictions among all actual positives. Important when the cost of false negatives is high.
    - Recall is the ratio of true positive predictions to the total number of actual positives (true positives + false negatives).
    - Recall = TP/(TP + FN) where
        - TP = True Positives (correctly predicted positive instances)
        - FN = False Negatives (actual positive instances that were incorrectly predicted as negative)
- F1-Score
    - The harmonic mean of precision and recall. It provides a single metric that balances both concerns.
    - F1-Score = 2 x Precision x Recall/(Precision + Recall)

These metrics can be calculated for each class in a multiclass classification problem and then averaged to get overall performance metrics (using methods such as macro, micro or weighted averages):

- Macro Average: Calculates metrics for each class independently and then takes the average. Treats all classes equally.
- Micro Average: Aggregates the contributions of all classes to compute the average metric. Gives more weight to the majority class.
- Weighted Average: Calculates metrics for each class and averages them, weighted by the number of true instances for each class. This accounts for class imbalance.

---

9. Plotting images from dataset

If we don't understand what our data looks like, we cannot build a good AI model. Hence, we plot some sample images from the dataset.

1. Plotting one image from each emotion folder

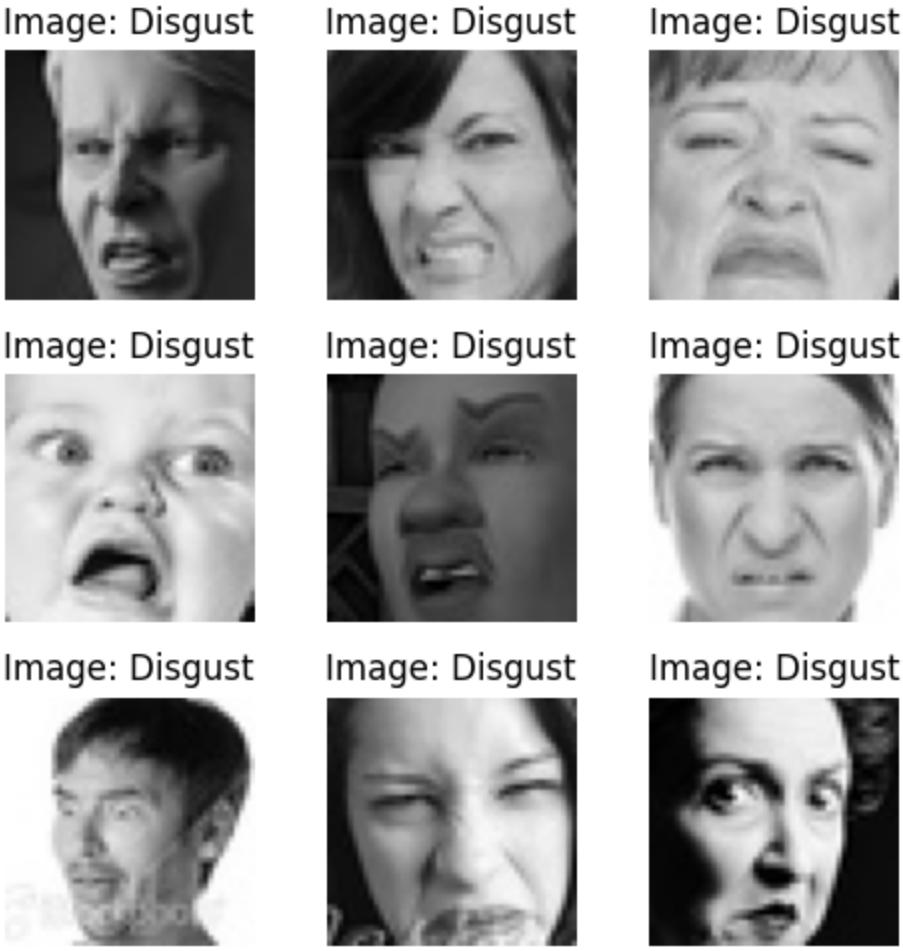
In [ ]:
```
emotions = os.listdir(train_dir)
plt.figure(figsize=(15,10))

for i, emotion in enumerate(emotions, 1):
    folder = os.path.join(train_dir, emotion)
    img_path = os.path.join(folder, os.listdir(folder)[42])
    img = plt.imread(img_path)
    plt.subplot(3, 4, i)
    plt.imshow(img, cmap='gray')
    plt.title(emotion)
    plt.axis('off')
```

2. Plotting random images from given emotion folder

In [ ]:
```
def plot_images_from_directory(directory_path, class_name, num_images=9):
    # Retrieve list of all file names in the directory
    image_filenames = os.listdir(directory_path)

    # If there are fewer images than requested, we'll just show them all
    if len(image_filenames) < num_images:
        print(f"Only found {len(image_filenames)} images in {directory_path}, displaying
        num_images = len(image_filenames)

    # Randomly select 'num_images' number of file names
    selected_images = random.sample(image_filenames, num_images)

    # Plotting the images
    fig, axes = plt.subplots(3, 3, figsize=(5, 5))  # Adjust the size as needed
    axes = axes.ravel()

    for i, image_file in enumerate(selected_images):
        image_path = os.path.join(directory_path, image_file)
        # image = Image.open(image_path)
        image = load_img(image_path)
        axes[i].imshow(image)
        axes[i].set_title(f"Image: {class_name}")
        axes[i].axis('off')  # Hide the axis

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

In [ ]:
```
# Plot random images from angry folder
angry_directory_path = '/content/train/angry'  # Replace with your directory path
plot_images_from_directory(angry_directory_path, class_name = 'Angry')
```

In [ ]:
```
# Plot random images from disgust folder
disgust_directory_path = '/content/train/disgust'  # Replace with your directory path
plot_images_from_directory(disgust_directory_path, class_name = 'Disgust')
```



In [ ]:
```
# Plot random images from surprise folder
surprise_directory_path = '/content/train/surprise'  # Replace with your directory path
plot_images_from_directory(surprise_directory_path, class_name = 'Surprise')
```

10. Checking image shapes and channels

In [ ]:
```
image = '/content/train/angry/Training_10118481.jpg'

import cv2

img = cv2.imread(image)  # Even though our images are grayscale, OpenCV by default loads

# If the image is loaded successfully, print its shape
if img is not None:
    print("Image shape:", img.shape)
else:
    print("The image could not be loaded. Please check the path and file permissions.")
```

```
Image shape: (48, 48, 3)
```

We need to load our images are grayscale itself, not as RGB. Hence, we use `cv2.IMREAD_GRAYSCALE` flag. This is a flag used in the OpenCV library to load an image in grayscale mode. When you use this flag with the `cv2.imread` function, it reads the image as a single-channel grayscale image, regardless of whether the original image is colored or not.

In [ ]:
```
import cv2

image_path = '/content/train/angry/Training_10118481.jpg'

# Load the image in grayscale
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
# If the image is loaded successfully, print its shape
if img is not None:
    print("Image shape:", img.shape)  # This should now print (48, 48)
else:
    print("The image could not be loaded. Please check the path and file permissions.")
```

```
Image shape: (48, 48)
```

---

11. Model 1 : Custom CNN from Scratch

1. Data Loading: Load Images using `Keras ImageDataGenerator`

Lets say we have 10000 images of 5 MB each in our dataset. Let our machine has a GPU with 5 GB memory. Since 10000 $\times$ 5 MB = 50 GB > 5 GB, we cannot load all images at once. Hence, we use generators to load images in batches. Lets say we use a batch size of 100. Then, generators load 10000 images in 100 batches of 100 images each. Since 100 $\times$ 5 MB = 0.5 GB < 5 GB, this is feasible.

Here, we use `Keras ImageDataGenerator` for loading images in batches. Since our images are inside a directory, we use `flow_from_directory` method of `Keras ImageDataGenerator` to create `train_generator`, `validation_generator` and `test_generator`.

In [ ]:
```
# Define paths to the train and validation directories
train_data_dir = '/content/train'
test_data_dir = '/content/test'
```

In [ ]:
```
# Set some parameters
img_width, img_height = 48, 48  # Size of images
batch_size = 64  # Can adjust based on your memory constraints
epochs = 10
num_classes = 7  # Update this based on the number of your classes
```

In [ ]:
```
# All pixel values will be rescaled by 1./255 (convert pixel values from [0, 255] to [0,
data_generator = ImageDataGenerator(rescale=1./255,
                                    validation_split=0.2)

# Automatically retrieve images and their classes for train and validation sets
train_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',  # Since Multi-class Classification (Use 'binary' for Bina
    color_mode='grayscale',
    subset='training')

validation_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale',
    subset='validation')

test_generator = data_generator.flow_from_directory(
    test_data_dir,
```

```
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale')
```

```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

2. Getting Class Labels

We need to know the class labels assigned to various emotions by the `Keras ImageDataGenerator`.

In [ ]:
```
# Accessing class labels for the training data
train_class_labels = train_generator.class_indices
print(f"Training class labels: {train_class_labels}")

# Accessing class labels for the validation data
validation_class_labels = validation_generator.class_indices
print(f"Validation class labels: {validation_class_labels}")

# Accessing class labels for the test data
test_class_labels = test_generator.class_indices
print(f"Test class labels: {test_class_labels}")
```

```
Training class labels: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4,
'sad': 5, 'surprise': 6}
Validation class labels: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4,
'sad': 5, 'surprise': 6}
Test class labels: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4, 'sa
d': 5, 'surprise': 6}
```

3. Building Model

In [ ]:
```
# Initialize the CNN
model = Sequential()

# CNN portion of the model responsible for feature extraction
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(img_width, img_hei
model.add(Activation('relu'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
model.add(Activation('relu'))
model.add(Conv2D(256, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(512, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
model.add(Activation('relu'))
model.add(Conv2D(512, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
# Flattening
model.add(Flatten())

# MLP (Multi-Layer Perceptron) portion of the model responsible for classification
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.25))
# Output layer
model.add(Dense(num_classes))
model.add(Activation('softmax'))  # Softmax activation for Multi-class Classification
```

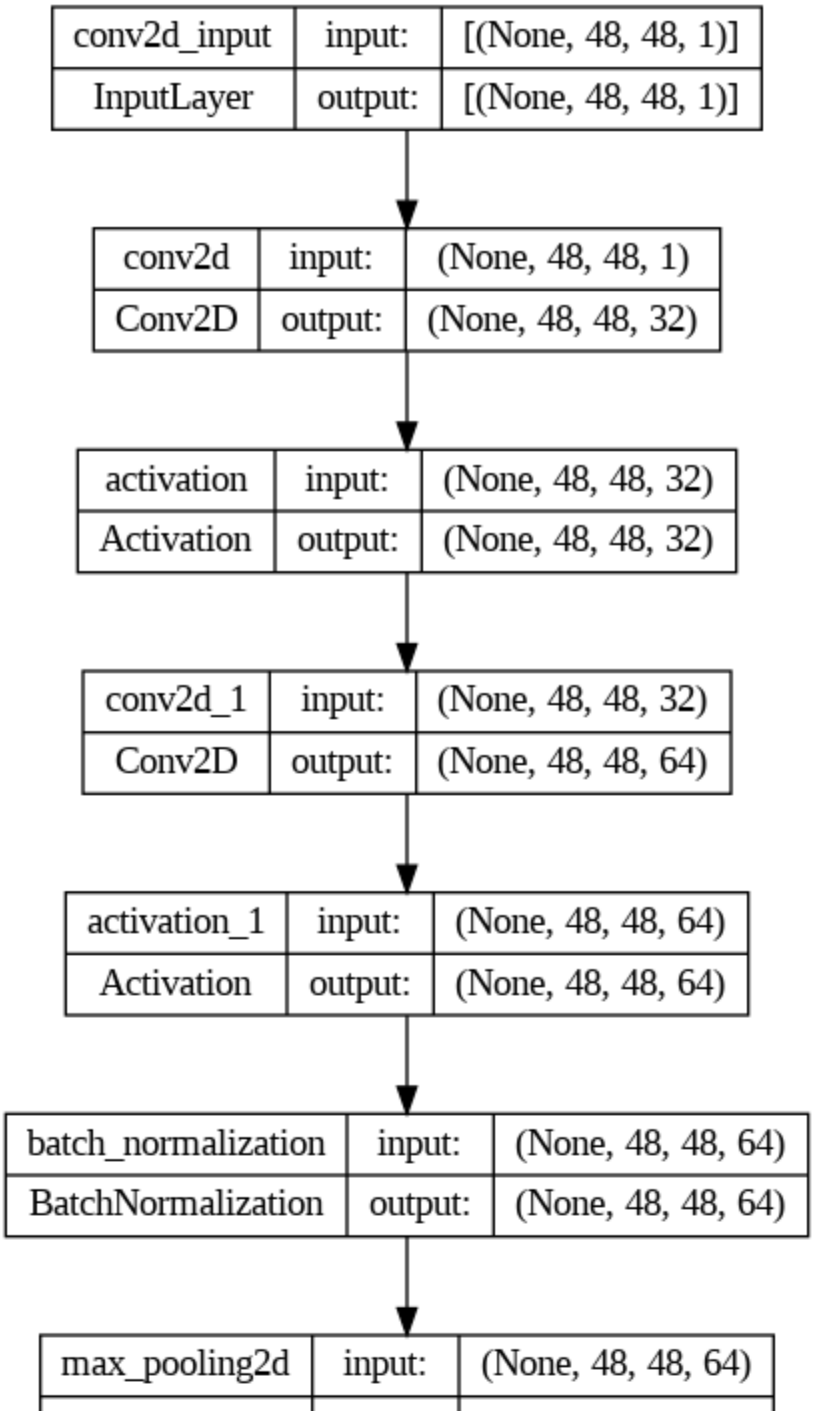
In [ ]:
```
model.summary()
```

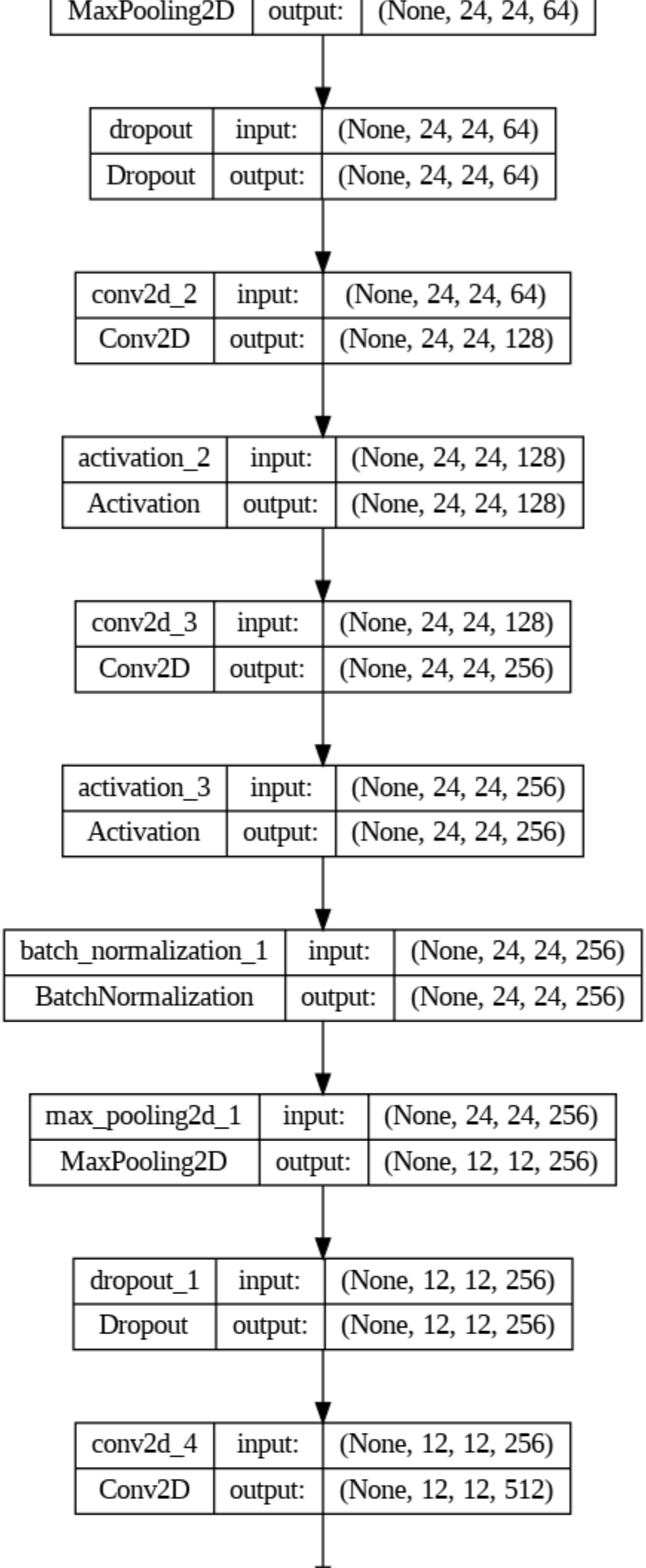
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 32)        320

 activation (Activation)     (None, 48, 48, 32)        0

 conv2d_1 (Conv2D)           (None, 48, 48, 64)        18496

 activation_1 (Activation)   (None, 48, 48, 64)        0

 batch_normalization (Batch  (None, 48, 48, 64)        256
 Normalization)

 max_pooling2d (MaxPooling2  (None, 24, 24, 64)        0
 D)

 dropout (Dropout)           (None, 24, 24, 64)        0

 conv2d_2 (Conv2D)           (None, 24, 24, 128)       73856

 activation_2 (Activation)   (None, 24, 24, 128)       0

 conv2d_3 (Conv2D)           (None, 24, 24, 256)       295168

 activation_3 (Activation)   (None, 24, 24, 256)       0

 batch_normalization_1 (Bat  (None, 24, 24, 256)       1024
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 256)       0
 g2D)

 dropout_1 (Dropout)         (None, 12, 12, 256)       0

 conv2d_4 (Conv2D)           (None, 12, 12, 512)       1180160

 activation_4 (Activation)   (None, 12, 12, 512)       0

 conv2d_5 (Conv2D)           (None, 12, 12, 512)       2359808

 activation_5 (Activation)   (None, 12, 12, 512)       0

 batch_normalization_2 (Bat  (None, 12, 12, 512)       2048
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 512)         0
 g2D)

 dropout_2 (Dropout)         (None, 6, 6, 512)         0
```

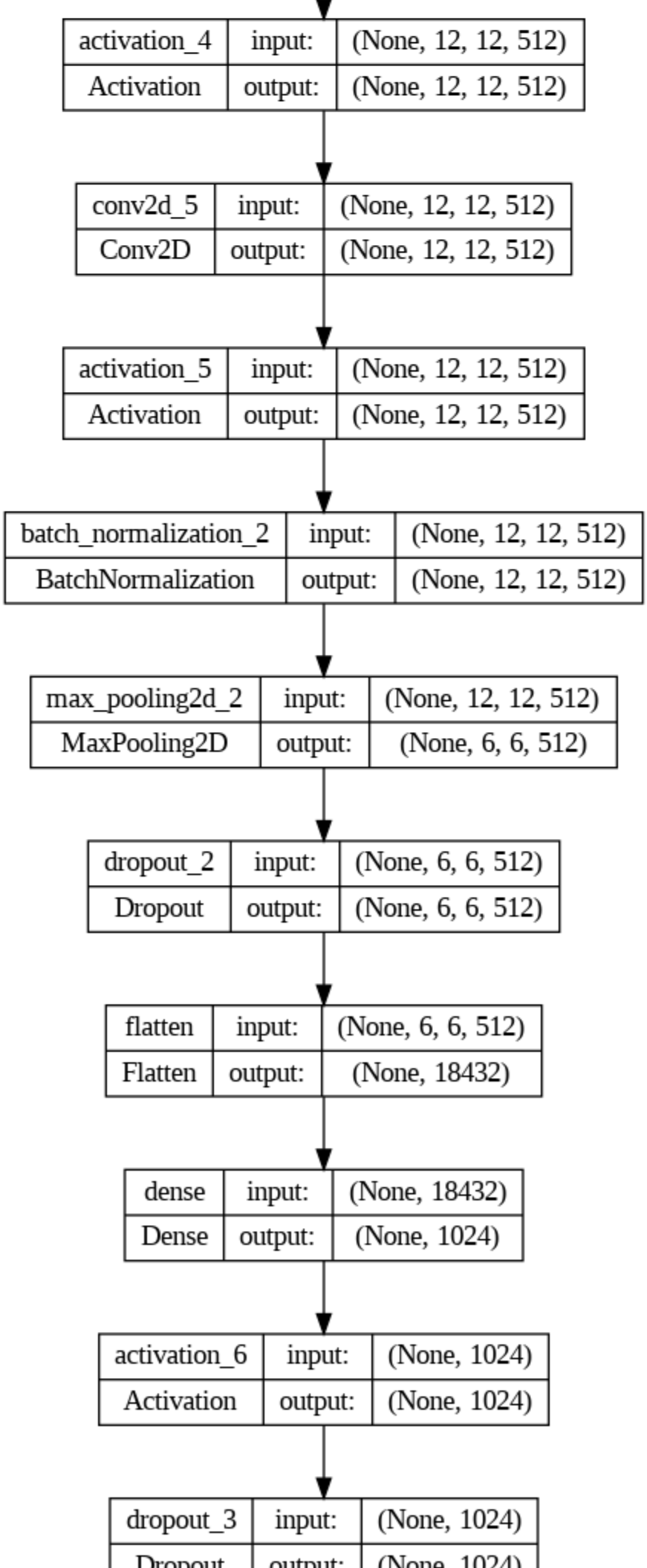
```
 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 1024)              18875392

 activation_6 (Activation)   (None, 1024)              0

 dropout_3 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 7)                 7175

 activation_7 (Activation)   (None, 7)                 0

=================================================================
Total params: 22813703 (87.03 MB)
Trainable params: 22812039 (87.02 MB)
Non-trainable params: 1664 (6.50 KB)
_________________________________________________________________
```

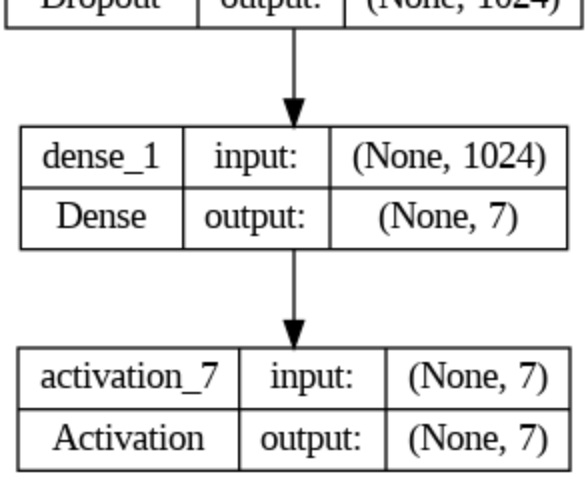
In [ ]:
```
# Plot model architecture
plot_model(model, to_file ='/content/FER_2013_Emotion_Classifier/Custom_CNN_From_Scratch
           show_shapes=True, show_layer_names=True)
```

Out[ ]:




MaxPooling2D output: (None, 24, 24, 64)
dropout input: (None, 24, 24, 64)
Dropout output: (None, 24, 24, 64)
conv2d_2 input: (None, 24, 24, 64)
Conv2D output: (None, 24, 24, 128)
activation_2 input: (None, 24, 24, 128)
Activation output: (None, 24, 24, 128)
conv2d_3 input: (None, 24, 24, 128)
Conv2D output: (None, 24, 24, 256)
activation_3 input: (None, 24, 24, 256)
Activation output: (None, 24, 24, 256)
batch_normalization_1 input: (None, 24, 24, 256)
BatchNormalization output: (None, 24, 24, 256)
max_pooling2d_1 input: (None, 24, 24, 256)
MaxPooling2D output: (None, 12, 12, 256)
dropout_1 input: (None, 12, 12, 256)
Dropout output: (None, 12, 12, 256)
conv2d_4 input: (None, 12, 12, 256)
Conv2D output: (None, 12, 12, 512)


activation_4 input: (None, 12, 12, 512)
Activation output: (None, 12, 12, 512)
conv2d_5 input: (None, 12, 12, 512)
Conv2D output: (None, 12, 12, 512)
activation_5 input: (None, 12, 12, 512)
Activation output: (None, 12, 12, 512)
batch_normalization_2 input: (None, 12, 12, 512)
BatchNormalization output: (None, 12, 12, 512)
max_pooling2d_2 input: (None, 12, 12, 512)
MaxPooling2D output: (None, 6, 6, 512)
dropout_2 input: (None, 6, 6, 512)
Dropout output: (None, 6, 6, 512)
flatten input: (None, 6, 6, 512)
Flatten output: (None, 18432)
dense input: (None, 18432)
Dense output: (None, 1024)
activation_6 input: (None, 1024)
Activation output: (None, 1024)
dropout_3 input: (None, 1024)



Note 1

During training, we will be supplying the model with a single batch of data at a time. Hence, `None` in the above output shapes will be replaced with the `batch_size` of the data loader (i.e. `Keras ImageDataGenerator`).

Note 2

Formulas for calculation of number of parameters for different types of layers is as below:

- Convolution layer:
  - Each filter in a Convolution layer has `filter size` $\times$ `no of channels` weights and a single bias.
  - No of parameters for a Convolution layer = (`filter size` $\times$ `no of channels` + 1) $\times$ `no of filters`
- Batch Normalization layer:
  - Each Batch Normalization layer has 4 parameters for each feature map (or channel) in the preceding layer:
    1. Gamma (scale factor): One parameter per channel
    2. Beta (shift factor): One parameter per channel
    3. Moving Mean: One parameter per channel
    4. Moving Variance: One parameter per channel
  - No of parameters for a Batch Normalization layer = `no of feature maps (or channels) in the preceding layer` $\times 4$

No of parameters for each layer is calculated as below:

- No of parameters in conv2d layer = $(3 \times 3 \times 1 + 1) \times 32 = 320$
- No of parameters in conv2d_1 layer = $(3 \times 3 \times 32 + 1) \times 64 = 18496$
- No of parameters in batch_normalization layer = $64 \times 4 = 256$
- No of parameters in conv2d_2 layer = $(3 \times 3 \times 64 + 1) \times 128 = 73856$
- No of parameters in conv2d_3 layer = $(3 \times 3 \times 128 + 1) \times 256 = 295168$
- No of parameters in batch_normalization_1 layer = $256 \times 4 = 1024$
- No of parameters in conv2d_4 layer = $(3 \times 3 \times 256 + 1) \times 512 = 1180160$
- No of parameters in conv2d_5 layer = $(3 \times 3 \times 512 + 1) \times 512 = 2359808$
- No of parameters in batch_normalization_2 layer = $512 \times 4 = 2048$
- No of parameters in dense layer = $(18432 \times 1024 + 1024) = 18875392$
- No of parameters in dense_1 layer = $(1024 \times 7 + 7) = 7175$

Total no of parameters = 22813703 $\implies$ 22813703 $\times$ 4 bytes = 89116 KB = 87.03 MB

4. Understanding Callbacks

1. `ModelCheckpoint` Callback

If training somehow got interrupted midway, we don't want to lose all the progress we have made training the model. We don't want to restart the training from the beginning. Hence every few epochs, we need to save the best model trained till that time. Keras `ModelCheckpoint` Callback can be used for this end.

We need to monitor validation loss and save the model every few epochs when validation loss is minimized. Hence, `monitor='val_loss'` and `mode='min'`. We save only the best model i.e. we save the best model trained till now every few epochs and we overwrite the same path whenever model gets even better. Hence, `save_best_only=True`.

In [ ]:
```
# File path for the model checkpoint
cnn_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_From_Scratch'
name = 'Custom_CNN_model.keras'
chk_path = os.path.join(cnn_path, name)
print(f"chk_path: {chk_path}")
```

```
chk_path: /content/FER_2013_Emotion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.
keras
```

In [ ]:
```
# Callback to save the model checkpoint
checkpoint = ModelCheckpoint(filepath=chk_path,
                             save_best_only=True,
                             verbose=1,
                             mode='min',
                             monitor='val_loss')
```

2. `EarlyStopping` Callback

Let us say our model is overfitting during training. If we do not stop the training, model is not learning anything new (it is simply learning the training data by heart and not generalising well). We are also wasting GPU resources and our precious time. If we are training our model for 1000 epochs, we cannot sit and monitor at what epoch we need to stop the training. Hence, we need to automate this task.

We can automate this task by monitoring validation loss and check whether it is not improving for a set number of epochs. Here, we need to stop the training if validation loss is not reducing (i.e. it is either staying the same or increasing) for 3 consecutive epochs. Hence, `min_delta=0` and `patience=3`. Upon stopping the training, we want to restore the best model saved by `ModelCheckpoint` callback. Hence, `restore_best_weights=True`.

In [ ]:
```
# Callback for early stopping
earlystop = EarlyStopping(monitor='val_loss',
                          min_delta=0,
                          patience=3,
                          verbose=1,
                          restore_best_weights=True)
```

3. `ReduceLROnPlateau` Callback

`ReduceLROnPlateau` callback reduces the learning rate when the model's performance stops improving. This can help the model converge more smoothly and potentially reach a better solution. We monitor validation loss and need to wait for 6 epochs to see an improvement in the monitored validation loss before reducing the learning rate. Hence, `monitor='val_loss'` and `patience=6`. If validation loss does not improve by at least 0.0001 in any of these 6 consecutive epochs, the learning rate is reduced to 20% of previous value. Hence, `factor=0.2` and `min_delta=0.0001`.

In [ ]:
```
# Callback to reduce learning rate
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=6,
                              verbose=1,
                              min_delta=0.0001)
```

4. `CSVLogger` Callback

In [ ]:
```
# Callback to log training data to a CSV file
csv_logger = CSVLogger(os.path.join(cnn_path,'training.log'))
```

In [ ]:
```
# Aggregating all callbacks into a list
callbacks = [checkpoint, earlystop, reduce_lr, csv_logger]  # Adjusted as per your use-c
```

5. Training Model

In [ ]:
```
# Compile the model
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [ ]:
```
# Calculate steps per epoch (no of batches needed to finish 1 epoch)
train_steps_per_epoch = np.ceil(train_generator.samples / train_generator.batch_size)
validation_steps_per_epoch = np.ceil(validation_generator.samples / validation_generator
test_steps_per_epoch = np.ceil(test_generator.samples / test_generator.batch_size)
print(f"train_steps_per_epoch = {train_steps_per_epoch}")
print(f"validation_steps_per_epoch = {validation_steps_per_epoch}")
print(f"test_steps_per_epoch = {test_steps_per_epoch}")
```

```
train_steps_per_epoch = 359.0
validation_steps_per_epoch = 90.0
test_steps_per_epoch = 113.0
```

In [ ]:
```
history = model.fit(
    train_generator,
    steps_per_epoch=train_steps_per_epoch,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_steps_per_epoch,
    callbacks=callbacks)
```

```
Epoch 1/10
359/359 [==============================] - ETA: 0s - loss: 12.2079 - accuracy: 0.2901
Epoch 1: val_loss improved from inf to 12.44419, saving model to /content/FER_2013_Emoti
on_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 44s 93ms/step - loss: 12.2079 - accuracy: 0.2
901 - val_loss: 12.4442 - val_accuracy: 0.2465 - lr: 1.0000e-04
Epoch 2/10
359/359 [==============================] - ETA: 0s - loss: 9.5819 - accuracy: 0.3722
```

```
Epoch 2: val_loss improved from 12.44419 to 8.68835, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 33s 91ms/step - loss: 9.5819 - accuracy: 0.37
22 - val_loss: 8.6883 - val_accuracy: 0.3933 - lr: 1.0000e-04
Epoch 3/10
359/359 [==============================] - ETA: 0s - loss: 7.3750 - accuracy: 0.4242
Epoch 3: val_loss improved from 8.68835 to 6.55143, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 28s 78ms/step - loss: 7.3750 - accuracy: 0.42
42 - val_loss: 6.5514 - val_accuracy: 0.4367 - lr: 1.0000e-04
Epoch 4/10
359/359 [==============================] - ETA: 0s - loss: 5.6164 - accuracy: 0.4682
Epoch 4: val_loss improved from 6.55143 to 5.13574, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 28s 78ms/step - loss: 5.6164 - accuracy: 0.46
82 - val_loss: 5.1357 - val_accuracy: 0.4423 - lr: 1.0000e-04
Epoch 5/10
359/359 [==============================] - ETA: 0s - loss: 4.3074 - accuracy: 0.5041
Epoch 5: val_loss improved from 5.13574 to 3.81375, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 28s 77ms/step - loss: 4.3074 - accuracy: 0.50
41 - val_loss: 3.8138 - val_accuracy: 0.5093 - lr: 1.0000e-04
Epoch 6/10
359/359 [==============================] - ETA: 0s - loss: 3.3649 - accuracy: 0.5417
Epoch 6: val_loss improved from 3.81375 to 3.07908, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 28s 78ms/step - loss: 3.3649 - accuracy: 0.54
17 - val_loss: 3.0791 - val_accuracy: 0.5309 - lr: 1.0000e-04
Epoch 7/10
359/359 [==============================] - ETA: 0s - loss: 2.7034 - accuracy: 0.5687
Epoch 7: val_loss improved from 3.07908 to 2.55015, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 28s 78ms/step - loss: 2.7034 - accuracy: 0.56
87 - val_loss: 2.5501 - val_accuracy: 0.5445 - lr: 1.0000e-04
Epoch 8/10
359/359 [==============================] - ETA: 0s - loss: 2.2257 - accuracy: 0.6038
Epoch 8: val_loss improved from 2.55015 to 2.20379, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 27s 76ms/step - loss: 2.2257 - accuracy: 0.60
38 - val_loss: 2.2038 - val_accuracy: 0.5560 - lr: 1.0000e-04
Epoch 9/10
359/359 [==============================] - ETA: 0s - loss: 1.8764 - accuracy: 0.6379
Epoch 9: val_loss improved from 2.20379 to 2.00547, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 27s 76ms/step - loss: 1.8764 - accuracy: 0.63
79 - val_loss: 2.0055 - val_accuracy: 0.5476 - lr: 1.0000e-04
Epoch 10/10
359/359 [==============================] - ETA: 0s - loss: 1.5953 - accuracy: 0.6790
Epoch 10: val_loss improved from 2.00547 to 1.79622, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_From_Scratch/Custom_CNN_model.keras
359/359 [==============================] - 30s 83ms/step - loss: 1.5953 - accuracy: 0.67
90 - val_loss: 1.7962 - val_accuracy: 0.5769 - lr: 1.0000e-04
```

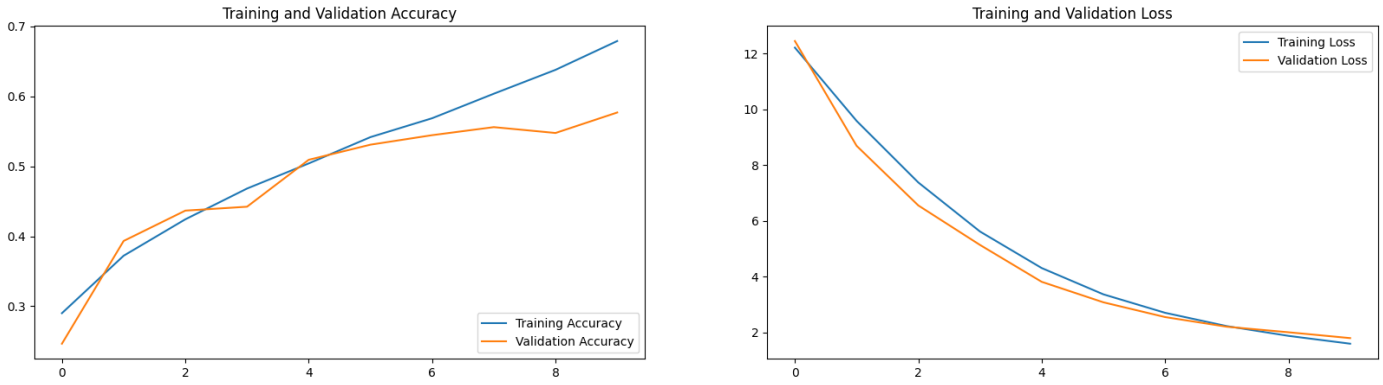
6. Plotting Performance Metrics

In [ ]:

```python
def plot_training_history(history, save_path=None):
    """
    Plots the training and validation accuracy and loss.

    Parameters:
    - history: A Keras History object. Contains the logs from the training process.

    Returns:
    - None. Displays the matplotlib plots for training/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 = range(len(acc))

    plt.figure(figsize=(20, 5))

    # Plot training and validation accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    # Plot training and validation loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')

    if save_path:
        plt.savefig(save_path)

    plt.show()
```

In [ ]:
```
training_history_save_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_From_Scrat
plot_training_history(history, training_history_save_path)
```



7. Model Evaluation

In [ ]:
```
train_loss, train_accu = model.evaluate(train_generator)
test_loss, test_accu = model.evaluate(test_generator)
print(f"Train accuracy = {train_accu*100:.2f} , Test accuracy = {test_accu*100:.2f}")
```

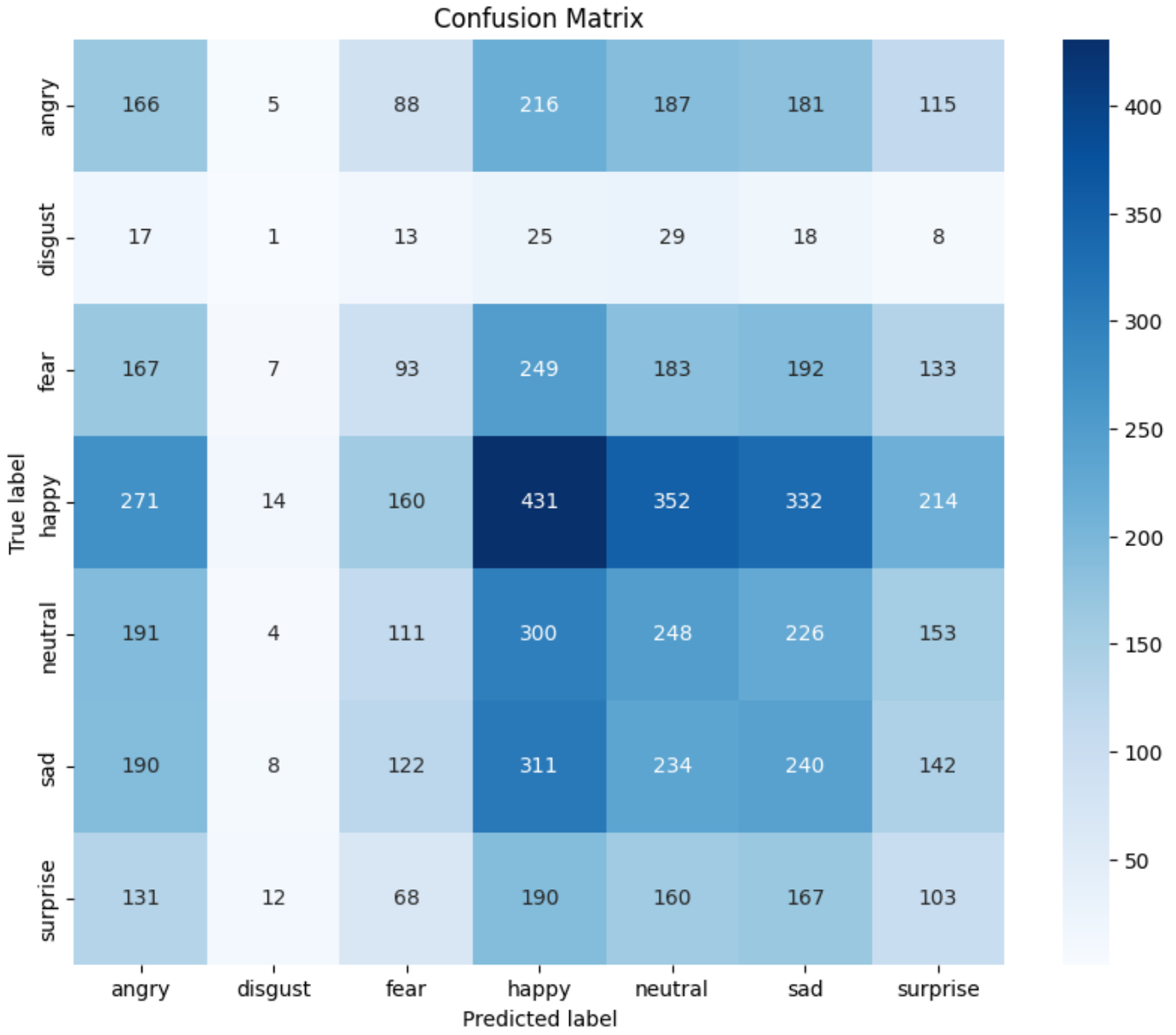
```
359/359 [==============================] - 9s 26ms/step - loss: 1.3237 - accuracy: 0.778
5
113/113 [==============================] - 3s 31ms/step - loss: 1.7721 - accuracy: 0.585
7
Train accuracy = 77.85 , Test accuracy = 58.57
```

8. Confusion Matrix

In [ ]:
```
# Assuming your true_classes and predicted_classes are already defined
true_classes = test_generator.classes
predicted_classes = np.argmax(model.predict(test_generator, steps=np.ceil(test_generator
```

```
class_labels = list(test_generator.class_indices.keys())

# Generate the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Plotting with seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')

confusion_matrix_save_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_From_Scrat
plt.savefig(confusion_matrix_save_path)

plt.show()
```

```
113/113 [==============================] - 5s 42ms/step
```



In the ideal scenario, we want the diagonal cells in the test data confusion matrix to be filled with very large values in comparison to the off-diagonal cells. Upon inspecting the above confusion matrix, we can see that our model is not performing well.

9. Classification Report

In [ ]:
```python
def save_classification_report(true_classes, predicted_classes, class_labels, save_path)
    """
    Generates a classification report and saves it as a PNG image.

    Parameters:
    - true_classes: Ground truth (correct) target values.
    - predicted_classes: Estimated targets as returned by a classifier.
    - class_labels: List of labels to index the matrix.
    - save_path: The path where the image will be saved.

    Returns:
    - None. Saves the classification report as a PNG image at the specified path.
    """
    report = classification_report(true_classes, predicted_classes, target_names=class_l

    # Create a matplotlib figure
    plt.figure(figsize=(10, 6))
    plt.text(0.01, 0.05, str(report), {'fontsize': 10}, fontproperties='monospace')
    plt.axis('off')

    # Save the figure
    plt.savefig(save_path, bbox_inches='tight', pad_inches=0.1)

    return report
```

In [ ]:
```python
# Get Classification Report
classification_report_save_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_From_
report = save_classification_report(true_classes, predicted_classes, class_labels, class
print(f"Classification Report:\n{report}")
```

```
Classification Report:
              precision    recall  f1-score   support

       angry       0.15      0.17      0.16       958
     disgust       0.02      0.01      0.01       111
        fear       0.14      0.09      0.11      1024
       happy       0.25      0.24      0.25      1774
     neutral       0.18      0.20      0.19      1233
         sad       0.18      0.19      0.18      1247
    surprise       0.12      0.12      0.12       831

    accuracy                           0.18      7178
   macro avg       0.15      0.15      0.15      7178
weighted avg       0.18      0.18      0.18      7178
```

	precision	recall	f1-score	support
angry	0.15	0.17	0.16	958
disgust	0.02	0.01	0.01	111
fear	0.14	0.09	0.11	1024
happy	0.25	0.24	0.25	1774
neutral	0.18	0.20	0.19	1233
sad	0.18	0.19	0.18	1247
surprise	0.12	0.12	0.12	831
accuracy			0.18	7178
macro avg	0.15	0.15	0.15	7178
weighted avg	0.18	0.18	0.18	7178

10. Making Predictions

In [ ]:

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

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

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_generator) - 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_generator[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_generator[Random_batch][1][Random_Img_Index[i]], a

    # Making a prediction using the model
    Model_Prediction = np.argmax(model.predict(tf.expand_dims(Random_Img, axis=0), verbo

    # 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_Predic
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classe

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

11. Conclusion

Even though the acccuracy is good enough (58.57 % on test data), we can see many misclassifications in the random test batch above. If the model is good, we will have very high values across diagonal cells of the test data confusion matrix in comparison to off-diagonal cells. Upon inspecting the Confusion Matrix, we can clearly see the discrepancy. We can also see the poor values of Precision, Recall and F1-Score from the Classification Report. Hence from all these observations, we can draw the conclusion that our model is not a good one.

---

12. Model 2 : Custom CNN with Image Augmentation

1. Data Loading: Load Images using `Keras ImageDataGenerator` with Data Augmentation

In [ ]:
```
# Define paths to the train and validation directories
train_data_dir = '/content/train'
test_data_dir = '/content/test'

# Set some parameters
img_width, img_height = 48, 48  # Size of images
batch_size = 64
epochs = 10
num_classes = 7  # Update this based on the number of your classes


# Initializing the ImageDataGenerator with data augmentation options for the training se
data_generator = ImageDataGenerator(
    rescale=1./255,  # Rescale the pixel values from [0, 255] to [0, 1]
    rotation_range=40,  # Degree range for random rotations
    width_shift_range=0.2,  # Range (as a fraction of total width) for random horizontal
```

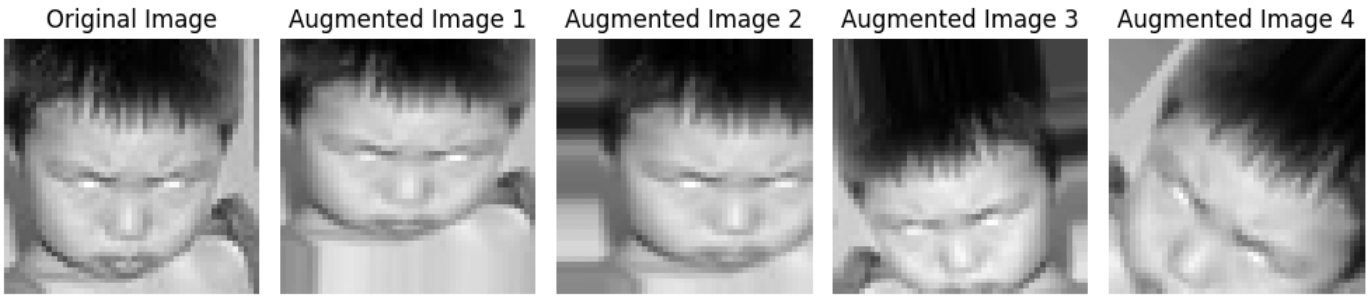
```
    height_shift_range=0.2,  # Range (as a fraction of total height) for random vertical
    shear_range=0.2,  # Shearing intensity (shear angle in counter-clockwise direction)
    zoom_range=0.2,  # Range for random zoom
    horizontal_flip=True,  # Randomly flip inputs horizontally
    fill_mode='nearest',  # Strategy to fill newly created pixels, which can appear afte
    validation_split=0.2  # Set the validation split; 20% of the data will be used for v
)

test_data_generator = ImageDataGenerator(rescale=1./255)


# Automatically retrieve images and their classes for train and validation sets
train_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale',
    subset='training')

validation_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale',
    subset='validation')

test_generator = test_data_generator.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale'
    )
```

```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

In [ ]:

```
# Path to your specific image
image_path = '/content/train/angry/Training_10118481.jpg'

# Load and convert the image to an array
img = load_img(image_path, color_mode='grayscale', target_size=(img_width, img_height))
img_array = img_to_array(img)  # Converts the image to a numpy array
img_array = img_array.reshape((1,) + img_array.shape)  # Reshapes the image to (1, width

# Setting up the plot
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(10, 4))
# Plot the original image first. Since img_array is a 3D array after reshaping, we use [
axes[0].imshow(img_array[0, :, :, 0], cmap='gray')
axes[0].set_title('Original Image')
axes[0].axis('off')

# Generate and plot augmented images
for i, ax in enumerate(axes.flat[1:]):  # Start from the second subplot
    # Generate a batch of augmented images
    aug_iter = data_generator.flow(img_array, batch_size=1)
    aug_img = next(aug_iter)[0]  # Get the first augmented image from the batch

    # Plot the augmented image. We directly use [:, :, 0] without an initial batch index
    ax.imshow(aug_img[:, :, 0], cmap='gray')
    ax.set_title(f'Augmented Image {i+1}')
    ax.axis('off')
```

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



2. Getting Class Labels

We need to know the class labels assigned to various emotions by the `Keras ImageDataGenerator`.

In [ ]:
```
# Accessing class labels for the training data
train_class_labels = train_generator.class_indices
print(f"Training class labels: {train_class_labels}")

# Accessing class labels for the validation data
validation_class_labels = validation_generator.class_indices
print(f"Validation class labels: {validation_class_labels}")

# Accessing class labels for the test data
test_class_labels = test_generator.class_indices
print(f"Test class labels: {test_class_labels}")
```

```
Training class labels: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4,
'sad': 5, 'surprise': 6}
Validation class labels: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4,
'sad': 5, 'surprise': 6}
Test class labels: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4, 'sa
d': 5, 'surprise': 6}
```

3. Building Model: Same CNN Model as above

In [ ]:
```
# Initialize the CNN
model = Sequential()

# CNN portion of the model responsible for feature extraction
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(img_width, img_hei
model.add(Activation('relu'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
model.add(Activation('relu'))
model.add(Conv2D(256, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(512, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
model.add(Activation('relu'))
model.add(Conv2D(512, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizer
```

```
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flattening
model.add(Flatten())

# MLP (Multi-Layer Perceptron) portion of the model responsible for classification
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.25))
# Output layer
model.add(Dense(num_classes))
model.add(Activation('softmax'))  # Softmax activation for Multi-class Classification
```

In [ ]:

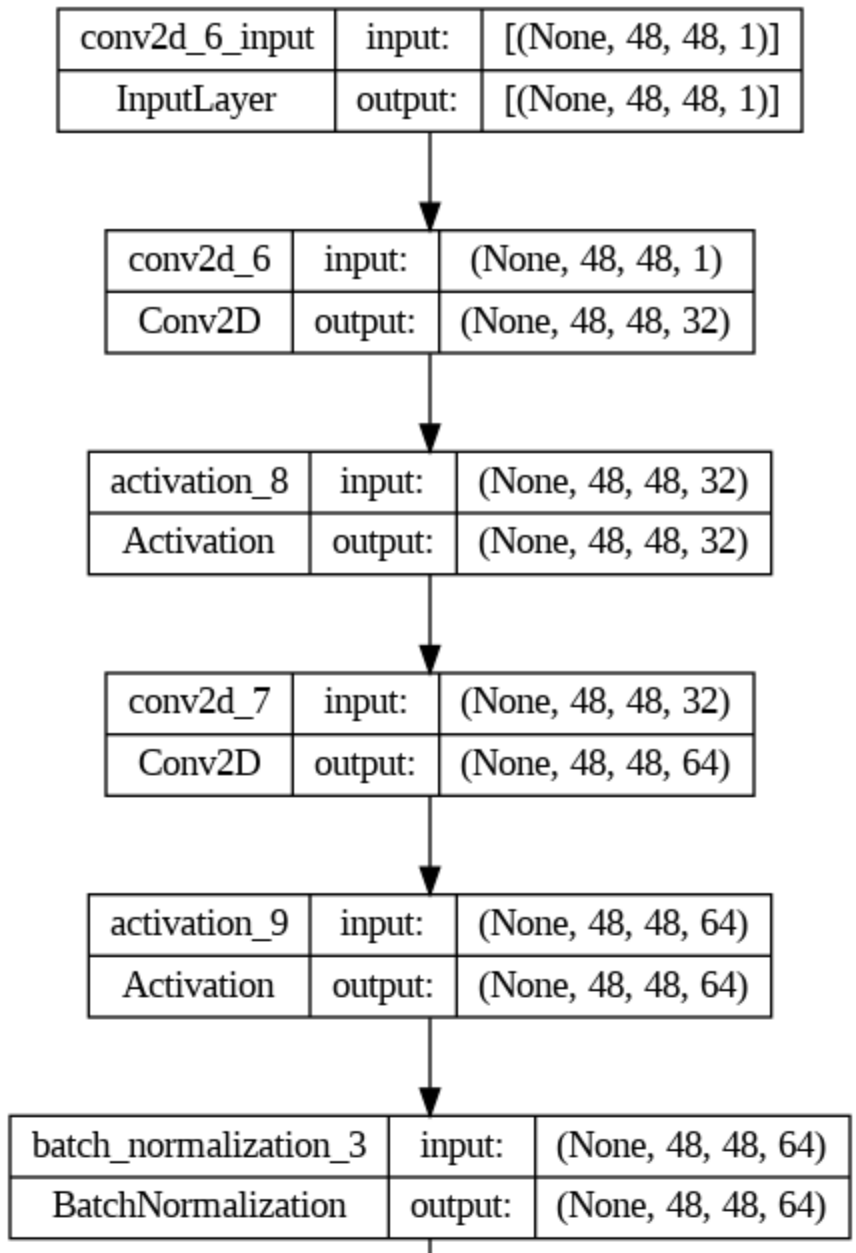
```
model.summary()
```

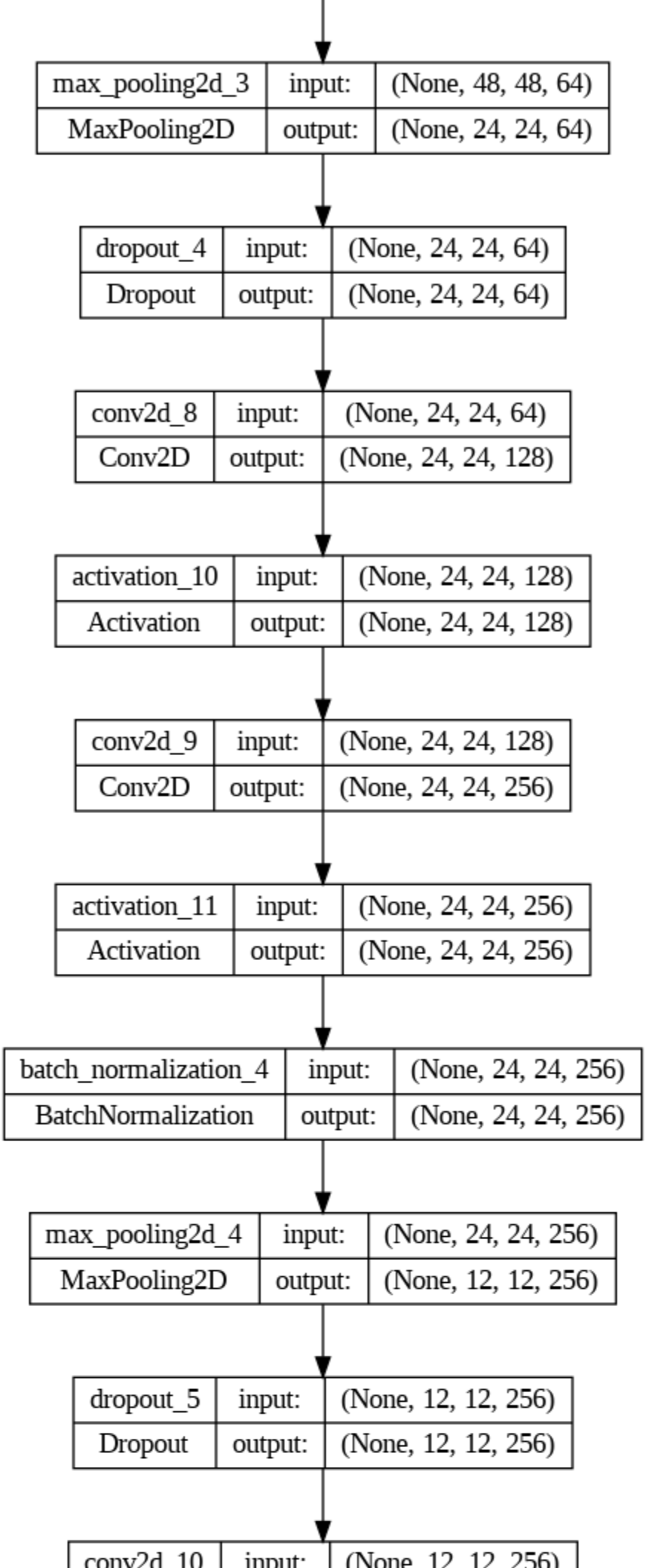
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 48, 48, 32)        320

 activation_8 (Activation)   (None, 48, 48, 32)        0

 conv2d_7 (Conv2D)           (None, 48, 48, 64)        18496

 activation_9 (Activation)   (None, 48, 48, 64)        0

 batch_normalization_3 (Bat  (None, 48, 48, 64)        256
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 24, 24, 64)        0
 g2D)

 dropout_4 (Dropout)         (None, 24, 24, 64)        0

 conv2d_8 (Conv2D)           (None, 24, 24, 128)       73856

 activation_10 (Activation)  (None, 24, 24, 128)       0

 conv2d_9 (Conv2D)           (None, 24, 24, 256)       295168

 activation_11 (Activation)  (None, 24, 24, 256)       0

 batch_normalization_4 (Bat  (None, 24, 24, 256)       1024
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 12, 12, 256)       0
 g2D)

 dropout_5 (Dropout)         (None, 12, 12, 256)       0

 conv2d_10 (Conv2D)          (None, 12, 12, 512)       1180160

 activation_12 (Activation)  (None, 12, 12, 512)       0

 conv2d_11 (Conv2D)          (None, 12, 12, 512)       2359808

 activation_13 (Activation)  (None, 12, 12, 512)       0

 batch_normalization_5 (Bat  (None, 12, 12, 512)       2048
 chNormalization)
```

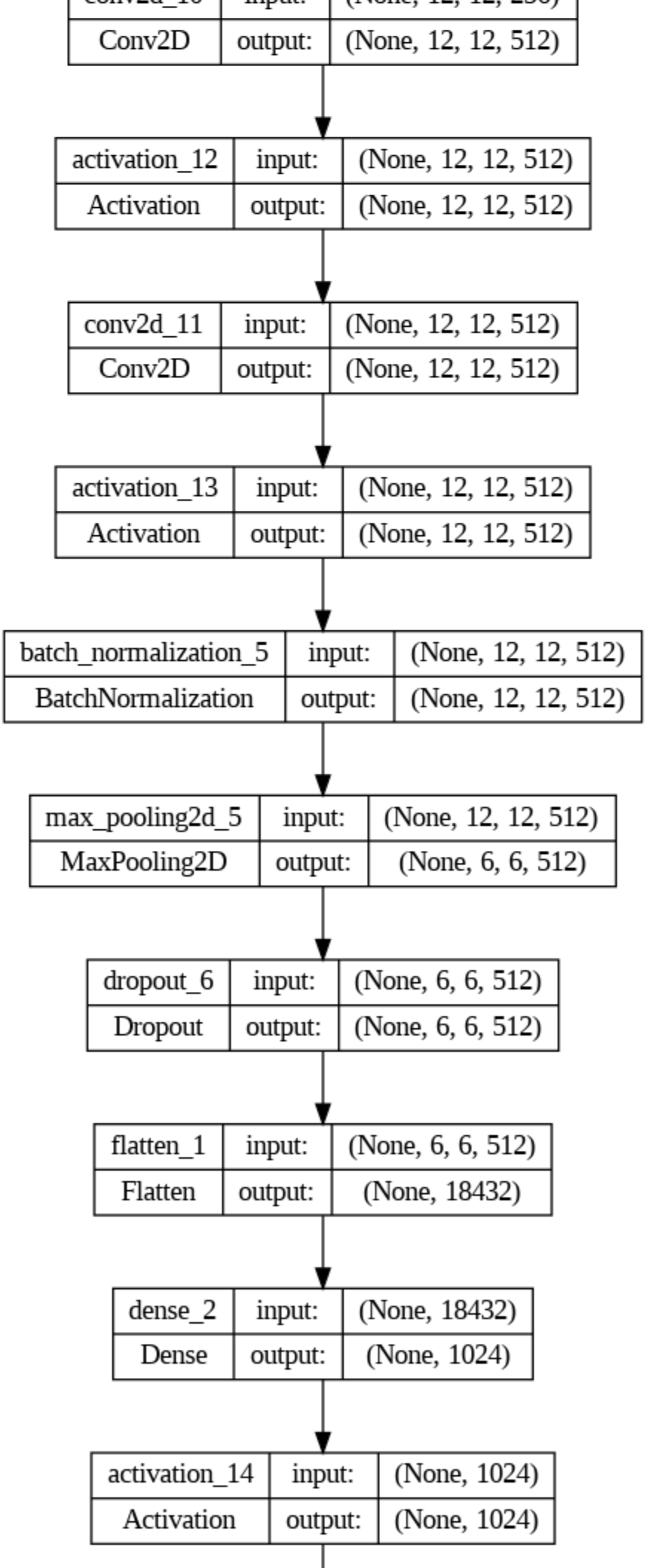
```
max_pooling2d_5 (MaxPoolin  (None, 6, 6, 512)         0
g2D)

dropout_6 (Dropout)         (None, 6, 6, 512)         0

flatten_1 (Flatten)         (None, 18432)             0

dense_2 (Dense)             (None, 1024)              18875392

activation_14 (Activation)  (None, 1024)              0

dropout_7 (Dropout)         (None, 1024)              0

dense_3 (Dense)             (None, 7)                 7175

activation_15 (Activation)  (None, 7)                 0

=================================================================
Total params: 22813703 (87.03 MB)
Trainable params: 22812039 (87.02 MB)
Non-trainable params: 1664 (6.50 KB)
_________________________________________________________________
```

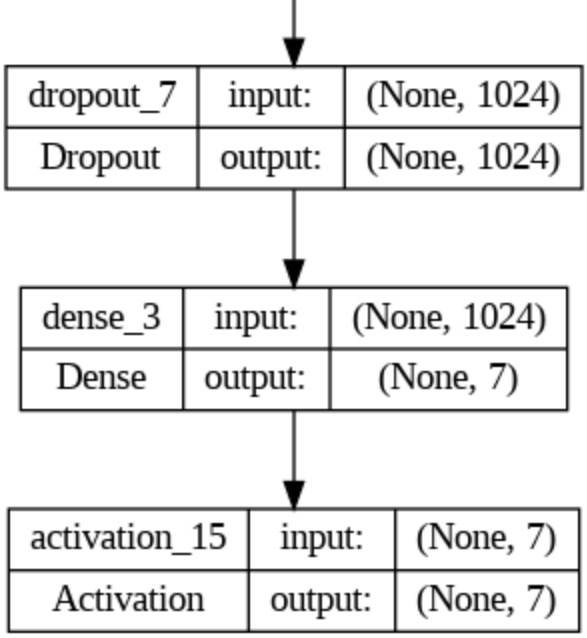
In [ ]:
```
# Plot model architecture
plot_model(model, to_file ='/content/FER_2013_Emotion_Classifier/Custom_CNN_With_Augment
           show_shapes=True, show_layer_names=True)
```

Out[ ]:




max_pooling2d_3 | input: | (None, 48, 48, 64)
MaxPooling2D | output: | (None, 24, 24, 64)
dropout_4 | input: | (None, 24, 24, 64)
Dropout | output: | (None, 24, 24, 64)
conv2d_8 | input: | (None, 24, 24, 64)
Conv2D | output: | (None, 24, 24, 128)
activation_10 | input: | (None, 24, 24, 128)
Activation | output: | (None, 24, 24, 128)
conv2d_9 | input: | (None, 24, 24, 128)
Conv2D | output: | (None, 24, 24, 256)
activation_11 | input: | (None, 24, 24, 256)
Activation | output: | (None, 24, 24, 256)
batch_normalization_4 | input: | (None, 24, 24, 256)
BatchNormalization | output: | (None, 24, 24, 256)
max_pooling2d_4 | input: | (None, 24, 24, 256)
MaxPooling2D | output: | (None, 12, 12, 256)
dropout_5 | input: | (None, 12, 12, 256)
Dropout | output: | (None, 12, 12, 256)


Conv2D output: (None, 12, 12, 512)
activation_12 input: (None, 12, 12, 512)
Activation output: (None, 12, 12, 512)
conv2d_11 input: (None, 12, 12, 512)
Conv2D output: (None, 12, 12, 512)
activation_13 input: (None, 12, 12, 512)
Activation output: (None, 12, 12, 512)
batch_normalization_5 input: (None, 12, 12, 512)
BatchNormalization output: (None, 12, 12, 512)
max_pooling2d_5 input: (None, 12, 12, 512)
MaxPooling2D output: (None, 6, 6, 512)
dropout_6 input: (None, 6, 6, 512)
Dropout output: (None, 6, 6, 512)
flatten_1 input: (None, 6, 6, 512)
Flatten output: (None, 18432)
dense_2 input: (None, 18432)
Dense output: (None, 1024)
activation_14 input: (None, 1024)
Activation output: (None, 1024)



4. Callbacks

In [ ]:
```
# File path for the model checkpoint
cnn_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_With_Augmentation'
name = 'Custom_CNN_augmented_model.keras'
chk_path = os.path.join(cnn_path, name)

# Callback to save the model checkpoint
checkpoint = ModelCheckpoint(filepath=chk_path,
                             save_best_only=True,
                             verbose=1,
                             mode='min',
                             monitor='val_loss')

# Callback for early stopping
earlystop = EarlyStopping(monitor='val_loss',
                          min_delta=0,
                          patience=3,
                          verbose=1,
                          restore_best_weights=True)

# Callback to reduce learning rate
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=6,
                              verbose=1,
                              min_delta=0.0001)

# Callback to log training data to a CSV file
csv_logger = CSVLogger(os.path.join(cnn_path,'training.log'))

# Aggregating all callbacks into a list
callbacks = [checkpoint, earlystop, reduce_lr, csv_logger]  # Adjusted as per your use-c
```

5. Training Model

In [ ]:
```
# Compile the model
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
```

```
    metrics=['accuracy']
)
```

In [ ]:
```
# Calculate steps per epoch (no of batches needed to finish 1 epoch)
train_steps_per_epoch = np.ceil(train_generator.samples / train_generator.batch_size)
validation_steps_per_epoch = np.ceil(validation_generator.samples / validation_generator
test_steps_per_epoch = np.ceil(test_generator.samples / test_generator.batch_size)
print(f"train_steps_per_epoch = {train_steps_per_epoch}")
print(f"validation_steps_per_epoch = {validation_steps_per_epoch}")
print(f"test_steps_per_epoch = {test_steps_per_epoch}")
```

```
train_steps_per_epoch = 359.0
validation_steps_per_epoch = 90.0
test_steps_per_epoch = 113.0
```

In [ ]:
```
history = model.fit(
    train_generator,
    steps_per_epoch=train_steps_per_epoch,
    epochs=100,
    validation_data=validation_generator,
    validation_steps=validation_steps_per_epoch,
    callbacks=callbacks)
```

```
Epoch 1/100
359/359 [==============================] - ETA: 0s - loss: 12.2480 - accuracy: 0.2273
Epoch 1: val_loss improved from inf to 12.48896, saving model to /content/FER_2013_Emoti
on_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 36s 89ms/step - loss: 12.2480 - accuracy: 0.2
273 - val_loss: 12.4890 - val_accuracy: 0.0782 - lr: 1.0000e-04
Epoch 2/100
359/359 [==============================] - ETA: 0s - loss: 9.6096 - accuracy: 0.2499
Epoch 2: val_loss improved from 12.48896 to 9.27839, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 33s 91ms/step - loss: 9.6096 - accuracy: 0.24
99 - val_loss: 9.2784 - val_accuracy: 0.2132 - lr: 1.0000e-04
Epoch 3/100
359/359 [==============================] - ETA: 0s - loss: 7.2824 - accuracy: 0.2610
Epoch 3: val_loss improved from 9.27839 to 6.38860, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 38s 106ms/step - loss: 7.2824 - accuracy: 0.2
610 - val_loss: 6.3886 - val_accuracy: 0.2594 - lr: 1.0000e-04
Epoch 4/100
359/359 [==============================] - ETA: 0s - loss: 5.4557 - accuracy: 0.2647
Epoch 4: val_loss improved from 6.38860 to 4.91571, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 32s 89ms/step - loss: 5.4557 - accuracy: 0.26
47 - val_loss: 4.9157 - val_accuracy: 0.2681 - lr: 1.0000e-04
Epoch 5/100
359/359 [==============================] - ETA: 0s - loss: 4.1465 - accuracy: 0.2735
Epoch 5: val_loss improved from 4.91571 to 4.03911, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 36s 99ms/step - loss: 4.1465 - accuracy: 0.27
35 - val_loss: 4.0391 - val_accuracy: 0.2670 - lr: 1.0000e-04
Epoch 6/100
359/359 [==============================] - ETA: 0s - loss: 3.2678 - accuracy: 0.2870
Epoch 6: val_loss improved from 4.03911 to 3.05481, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 42s 118ms/step - loss: 3.2678 - accuracy: 0.2
870 - val_loss: 3.0548 - val_accuracy: 0.2851 - lr: 1.0000e-04
Epoch 7/100
359/359 [==============================] - ETA: 0s - loss: 2.6780 - accuracy: 0.3208
Epoch 7: val_loss improved from 3.05481 to 2.50704, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 35s 97ms/step - loss: 2.6780 - accuracy: 0.32
08 - val_loss: 2.5070 - val_accuracy: 0.3245 - lr: 1.0000e-04
Epoch 8/100
```

```
359/359 [==============================] - ETA: 0s - loss: 2.2677 - accuracy: 0.3713
Epoch 8: val_loss improved from 2.50704 to 2.15582, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 44s 124ms/step - loss: 2.2677 - accuracy: 0.3
713 - val_loss: 2.1558 - val_accuracy: 0.3731 - lr: 1.0000e-04
Epoch 9/100
359/359 [==============================] - ETA: 0s - loss: 2.0011 - accuracy: 0.4097
Epoch 9: val_loss improved from 2.15582 to 1.90488, saving model to /content/FER_2013_Em
otion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 38s 106ms/step - loss: 2.0011 - accuracy: 0.4
097 - val_loss: 1.9049 - val_accuracy: 0.4180 - lr: 1.0000e-04
Epoch 10/100
359/359 [==============================] - ETA: 0s - loss: 1.8194 - accuracy: 0.4438
Epoch 10: val_loss improved from 1.90488 to 1.78520, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 32s 90ms/step - loss: 1.8194 - accuracy: 0.44
38 - val_loss: 1.7852 - val_accuracy: 0.4346 - lr: 1.0000e-04
Epoch 11/100
359/359 [==============================] - ETA: 0s - loss: 1.7024 - accuracy: 0.4629
Epoch 11: val_loss improved from 1.78520 to 1.67345, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 36s 100ms/step - loss: 1.7024 - accuracy: 0.4
629 - val_loss: 1.6735 - val_accuracy: 0.4588 - lr: 1.0000e-04
Epoch 12/100
359/359 [==============================] - ETA: 0s - loss: 1.6126 - accuracy: 0.4867
Epoch 12: val_loss improved from 1.67345 to 1.61659, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 34s 94ms/step - loss: 1.6126 - accuracy: 0.48
67 - val_loss: 1.6166 - val_accuracy: 0.4628 - lr: 1.0000e-04
Epoch 13/100
359/359 [==============================] - ETA: 0s - loss: 1.5530 - accuracy: 0.4965
Epoch 13: val_loss improved from 1.61659 to 1.51963, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 34s 94ms/step - loss: 1.5530 - accuracy: 0.49
65 - val_loss: 1.5196 - val_accuracy: 0.5027 - lr: 1.0000e-04
Epoch 14/100
359/359 [==============================] - ETA: 0s - loss: 1.4964 - accuracy: 0.5106
Epoch 14: val_loss improved from 1.51963 to 1.49480, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 36s 100ms/step - loss: 1.4964 - accuracy: 0.5
106 - val_loss: 1.4948 - val_accuracy: 0.5095 - lr: 1.0000e-04
Epoch 15/100
359/359 [==============================] - ETA: 0s - loss: 1.4618 - accuracy: 0.5200
Epoch 15: val_loss improved from 1.49480 to 1.47626, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 37s 103ms/step - loss: 1.4618 - accuracy: 0.5
200 - val_loss: 1.4763 - val_accuracy: 0.5198 - lr: 1.0000e-04
Epoch 16/100
359/359 [==============================] - ETA: 0s - loss: 1.4303 - accuracy: 0.5309
Epoch 16: val_loss improved from 1.47626 to 1.43422, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 32s 89ms/step - loss: 1.4303 - accuracy: 0.53
09 - val_loss: 1.4342 - val_accuracy: 0.5165 - lr: 1.0000e-04
Epoch 17/100
359/359 [==============================] - ETA: 0s - loss: 1.4033 - accuracy: 0.5360
Epoch 17: val_loss improved from 1.43422 to 1.41836, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 34s 93ms/step - loss: 1.4033 - accuracy: 0.53
60 - val_loss: 1.4184 - val_accuracy: 0.5203 - lr: 1.0000e-04
Epoch 18/100
359/359 [==============================] - ETA: 0s - loss: 1.3778 - accuracy: 0.5414
Epoch 18: val_loss improved from 1.41836 to 1.38394, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 38s 105ms/step - loss: 1.3778 - accuracy: 0.5
414 - val_loss: 1.3839 - val_accuracy: 0.5431 - lr: 1.0000e-04
Epoch 19/100
```

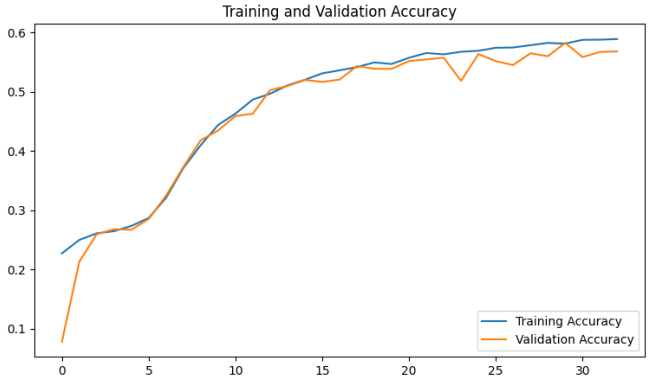
```
359/359 [==============================] - ETA: 0s - loss: 1.3601 - accuracy: 0.5493
Epoch 19: val_loss improved from 1.38394 to 1.36593, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 45s 125ms/step - loss: 1.3601 - accuracy: 0.5
493 - val_loss: 1.3659 - val_accuracy: 0.5386 - lr: 1.0000e-04
Epoch 20/100
359/359 [==============================] - ETA: 0s - loss: 1.3541 - accuracy: 0.5467
Epoch 20: val_loss did not improve from 1.36593
359/359 [==============================] - 30s 84ms/step - loss: 1.3541 - accuracy: 0.54
67 - val_loss: 1.3708 - val_accuracy: 0.5384 - lr: 1.0000e-04
Epoch 21/100
359/359 [==============================] - ETA: 0s - loss: 1.3342 - accuracy: 0.5572
Epoch 21: val_loss improved from 1.36593 to 1.35392, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 32s 89ms/step - loss: 1.3342 - accuracy: 0.55
72 - val_loss: 1.3539 - val_accuracy: 0.5516 - lr: 1.0000e-04
Epoch 22/100
359/359 [==============================] - ETA: 0s - loss: 1.3198 - accuracy: 0.5652
Epoch 22: val_loss improved from 1.35392 to 1.34944, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 33s 91ms/step - loss: 1.3198 - accuracy: 0.56
52 - val_loss: 1.3494 - val_accuracy: 0.5544 - lr: 1.0000e-04
Epoch 23/100
359/359 [==============================] - ETA: 0s - loss: 1.3144 - accuracy: 0.5629
Epoch 23: val_loss improved from 1.34944 to 1.32455, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 32s 90ms/step - loss: 1.3144 - accuracy: 0.56
29 - val_loss: 1.3245 - val_accuracy: 0.5572 - lr: 1.0000e-04
Epoch 24/100
359/359 [==============================] - ETA: 0s - loss: 1.2959 - accuracy: 0.5673
Epoch 24: val_loss did not improve from 1.32455
359/359 [==============================] - 30s 84ms/step - loss: 1.2959 - accuracy: 0.56
73 - val_loss: 1.4036 - val_accuracy: 0.5182 - lr: 1.0000e-04
Epoch 25/100
359/359 [==============================] - ETA: 0s - loss: 1.2871 - accuracy: 0.5690
Epoch 25: val_loss improved from 1.32455 to 1.31695, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 33s 93ms/step - loss: 1.2871 - accuracy: 0.56
90 - val_loss: 1.3170 - val_accuracy: 0.5633 - lr: 1.0000e-04
Epoch 26/100
359/359 [==============================] - ETA: 0s - loss: 1.2817 - accuracy: 0.5740
Epoch 26: val_loss did not improve from 1.31695
359/359 [==============================] - 31s 86ms/step - loss: 1.2817 - accuracy: 0.57
40 - val_loss: 1.3325 - val_accuracy: 0.5515 - lr: 1.0000e-04
Epoch 27/100
359/359 [==============================] - ETA: 0s - loss: 1.2764 - accuracy: 0.5745
Epoch 27: val_loss did not improve from 1.31695
359/359 [==============================] - 32s 88ms/step - loss: 1.2764 - accuracy: 0.57
45 - val_loss: 1.3393 - val_accuracy: 0.5449 - lr: 1.0000e-04
Epoch 28/100
359/359 [==============================] - ETA: 0s - loss: 1.2648 - accuracy: 0.5783
Epoch 28: val_loss improved from 1.31695 to 1.28761, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 38s 105ms/step - loss: 1.2648 - accuracy: 0.5
783 - val_loss: 1.2876 - val_accuracy: 0.5647 - lr: 1.0000e-04
Epoch 29/100
359/359 [==============================] - ETA: 0s - loss: 1.2558 - accuracy: 0.5822
Epoch 29: val_loss did not improve from 1.28761
359/359 [==============================] - 31s 85ms/step - loss: 1.2558 - accuracy: 0.58
22 - val_loss: 1.2969 - val_accuracy: 0.5597 - lr: 1.0000e-04
Epoch 30/100
359/359 [==============================] - ETA: 0s - loss: 1.2559 - accuracy: 0.5810
Epoch 30: val_loss improved from 1.28761 to 1.27221, saving model to /content/FER_2013_E
motion_Classifier/Custom_CNN_With_Augmentation/Custom_CNN_augmented_model.keras
359/359 [==============================] - 32s 90ms/step - loss: 1.2559 - accuracy: 0.58
10 - val_loss: 1.2722 - val_accuracy: 0.5823 - lr: 1.0000e-04
```

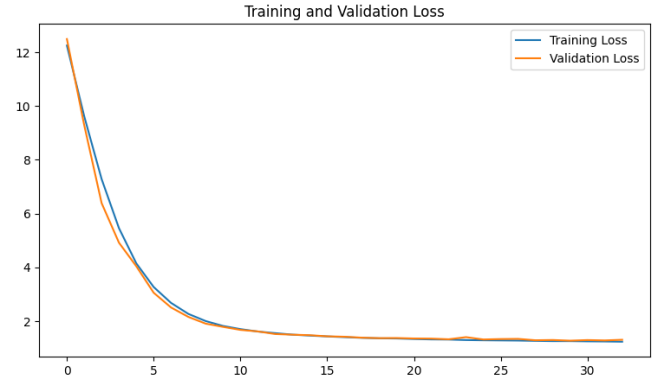
```
Epoch 31/100
359/359 [==============================] - ETA: 0s - loss: 1.2476 - accuracy: 0.5874
Epoch 31: val_loss did not improve from 1.27221
359/359 [==============================] - 31s 87ms/step - loss: 1.2476 - accuracy: 0.58
74 - val_loss: 1.2943 - val_accuracy: 0.5583 - lr: 1.0000e-04
Epoch 32/100
359/359 [==============================] - ETA: 0s - loss: 1.2425 - accuracy: 0.5876
Epoch 32: val_loss did not improve from 1.27221
359/359 [==============================] - 31s 86ms/step - loss: 1.2425 - accuracy: 0.58
76 - val_loss: 1.2814 - val_accuracy: 0.5670 - lr: 1.0000e-04
Epoch 33/100
359/359 [==============================] - ETA: 0s - loss: 1.2340 - accuracy: 0.5887
Epoch 33: val_loss did not improve from 1.27221
Restoring model weights from the end of the best epoch: 30.
359/359 [==============================] - 30s 84ms/step - loss: 1.2340 - accuracy: 0.58
87 - val_loss: 1.3059 - val_accuracy: 0.5678 - lr: 1.0000e-04
Epoch 33: early stopping
```

6. Plotting Performance Metrics

In [ ]:
```python
def plot_training_history(history, save_path=None):
    """
    Plots the training and validation accuracy and loss.

    Parameters:
    - history: A Keras History object. Contains the logs from the training process.

    Returns:
    - None. Displays the matplotlib plots for training/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 = range(len(acc))

    plt.figure(figsize=(20, 5))

    # Plot training and validation accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    # Plot training and validation loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')

    if save_path:
        plt.savefig(save_path)

    plt.show()
```

In [ ]:
```python
training_history_save_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_With_Augme
plot_training_history(history, training_history_save_path)
```





7. Model Evaluation

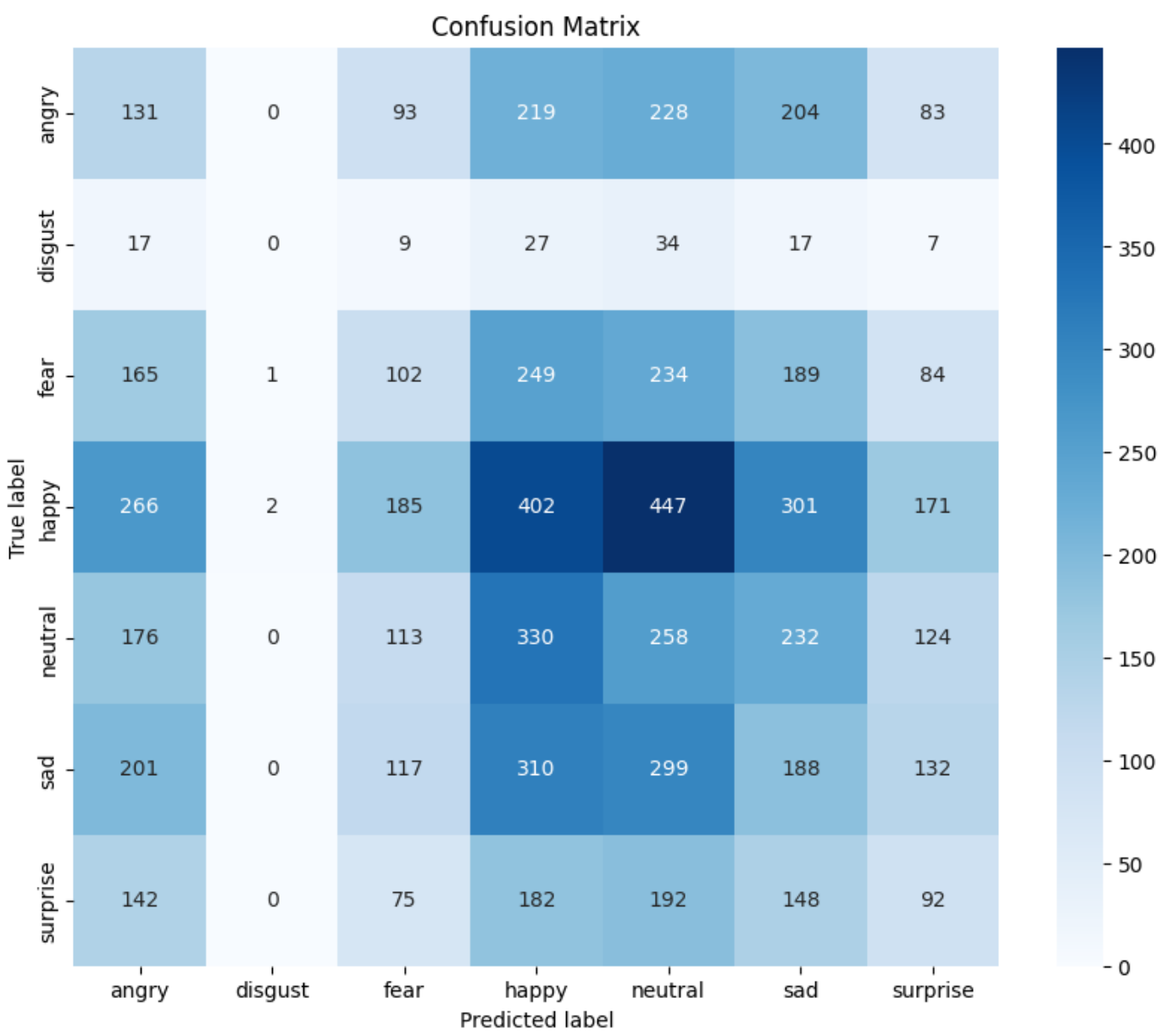
In [ ]:
```
train_loss, train_accu = model.evaluate(train_generator)
test_loss, test_accu = model.evaluate(test_generator)
print(f"Train accuracy = {train_accu*100:.2f} , Test accuracy = {test_accu*100:.2f}")
```

```
359/359 [==============================] - 21s 58ms/step - loss: 1.2024 - accuracy: 0.60
00
113/113 [==============================] - 3s 24ms/step - loss: 1.1852 - accuracy: 0.610
9
Train accuracy = 60.00 , Test accuracy = 61.09
```

8. Confusion Matrix

In [ ]:
```
# Assuming your true_classes and predicted_classes are already defined
true_classes = test_generator.classes
predicted_classes = np.argmax(model.predict(test_generator, steps=np.ceil(test_generator
class_labels = list(test_generator.class_indices.keys())

# Generate the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Plotting with seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')

confusion_matrix_save_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_With_Augme
plt.savefig(confusion_matrix_save_path)

plt.show()
```

```
113/113 [==============================] - 6s 46ms/step
```



9. Classification Report

In [ ]:

```python
def save_classification_report(true_classes, predicted_classes, class_labels, save_path)
    """
    Generates a classification report and saves it as a PNG image.

    Parameters:
    - true_classes: Ground truth (correct) target values.
    - predicted_classes: Estimated targets as returned by a classifier.
    - class_labels: List of labels to index the matrix.
    - save_path: The path where the image will be saved.

    Returns:
    - None. Saves the classification report as a PNG image at the specified path.
    """
    report = classification_report(true_classes, predicted_classes, target_names=class_l

    # Create a matplotlib figure
    plt.figure(figsize=(10, 6))
    plt.text(0.01, 0.05, str(report), {'fontsize': 10}, fontproperties='monospace')
    plt.axis('off')

    # Save the figure
    plt.savefig(save_path, bbox_inches='tight', pad_inches=0.1)
```

```
    return report
```

In [ ]:
```
# Get Classification Report
classification_report_save_path = '/content/FER_2013_Emotion_Classifier/Custom_CNN_With_
report = save_classification_report(true_classes, predicted_classes, class_labels, class
print(f"Classification Report:\n{report}")
```

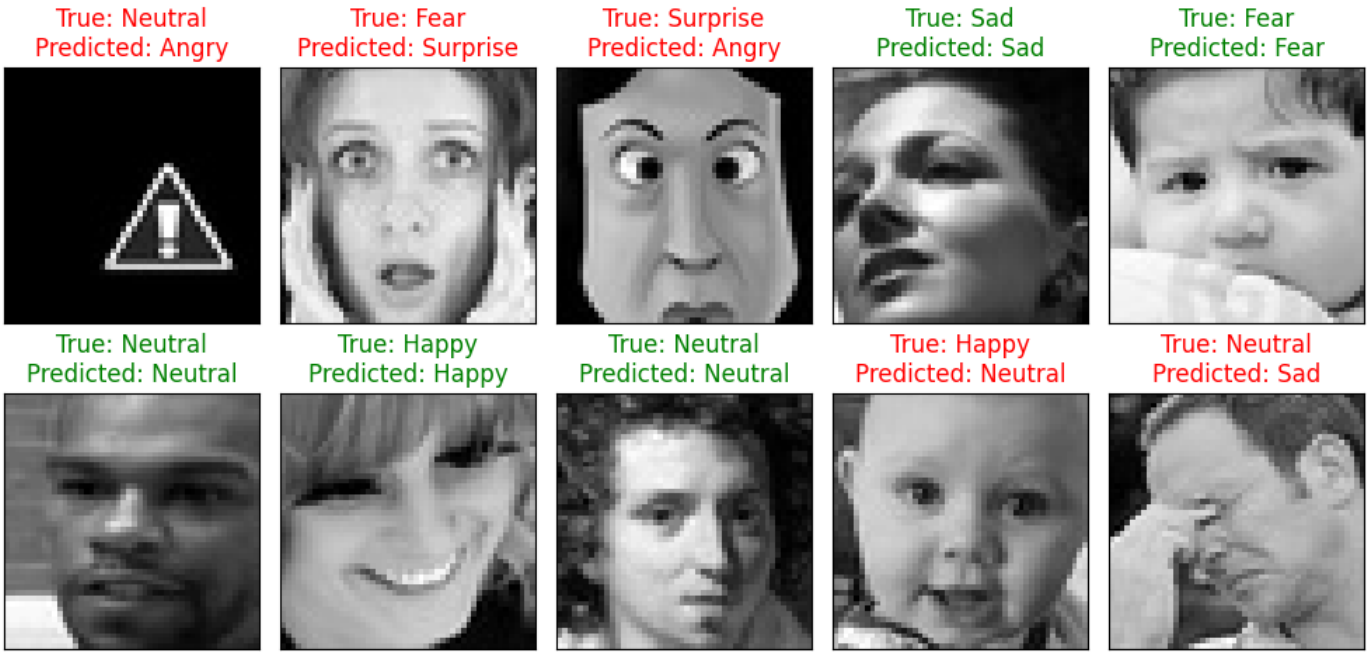
```
Classification Report:
              precision    recall  f1-score   support

       angry       0.12      0.14      0.13       958
     disgust       0.00      0.00      0.00       111
        fear       0.15      0.10      0.12      1024
       happy       0.23      0.23      0.23      1774
     neutral       0.15      0.21      0.18      1233
         sad       0.15      0.15      0.15      1247
    surprise       0.13      0.11      0.12       831

    accuracy                           0.16      7178
   macro avg       0.13      0.13      0.13      7178
weighted avg       0.16      0.16      0.16      7178
```


```
              precision    recall  f1-score   support

       angry       0.12      0.14      0.13       958
     disgust       0.00      0.00      0.00       111
        fear       0.15      0.10      0.12      1024
       happy       0.23      0.23      0.23      1774
     neutral       0.15      0.21      0.18      1233
         sad       0.15      0.15      0.15      1247
    surprise       0.13      0.11      0.12       831

    accuracy                           0.16      7178
   macro avg       0.13      0.13      0.13      7178
weighted avg       0.16      0.16      0.16      7178
```


10. Making Predictions

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

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

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_generator) - 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_generator[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_generator[Random_batch][1][Random_Img_Index[i]], a

    # Making a prediction using the model
    Model_Prediction = np.argmax(model.predict(tf.expand_dims(Random_Img, axis=0), verbo

    # 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_Predic
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classe

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



11. Conclusion

Even though the acccuracy is good enough (61.09 % on test data), we can see many misclassifications in the random test batch above. If the model is good, we will have very high values across diagonal cells of the test data confusion matrix in comparison to off-diagonal cells. Upon inspecting the Confusion Matrix, we can clearly see the discrepancy. We can also see the poor values of Precision, Recall and F1-Score from the Classification Report. Hence from all these observations, we can draw the conclusion that our model is not a good one. Even Data Augmentation is not yielding good results.

---

13. Transfer Learning and Fine-tuning

1. Background

- Problems with training our own model
    - Deep Learning is very costly since it requires
        - lots of data
        - lots of time for training
        - lots of GPU compute
    - No specific way to get best performing model for our dataset

      Ambiguity with regards to

        - Which kernel size to take ?
        - How many kernels per layer ?
        - How many layers ?
        - Which activation to use ?
- Solution:
    - Scientists did a lot of experiments with lots of data and with lots of hyperparameter choices (kernel size, no of kernels, no of layers, activation etc.) and came up with Deep Learning models that have very high accuracy on gigantic datasets.
    - ILSVRC (ImageNet Large Scale Visual Recognition Challenge):
        - ImageNet dataset has 1.4 Million images in 1000 classes
        - Build a Deep Learning model that has the most accuracy on ImageNet dataset
    - If this Deep Learning model performs really well in a gigantic dataset like ImageNet with more no of images and more no of classes, why can't we use this for our dataset (of course with less no of images and less no of classes) ?

2. Transfer Learning

- Technique that focuses on storing knowledge gained while solving one problem and applying it to a different problem
- With Transfer Learning, we can train a Deep Learning model faster in comparison to training the model from scratch. This is because the pretrained model has already learned the low level features really well. These low level features are common for any Computer Vision task. Hence, we just need to fine tune the high level features of the pretrained model for our dataset while keeping the low level features intact.

3. Transfer Learning in Action

We can do Transfer Learning in two ways:

- When our dataset is similar to ImageNet dataset:
    1. Freeze all Convolution layers
    2. Modify and fine-tune Fully Connected layers for our dataset
        - For example, if our dataset has 10 classes, we need to use a Fully Connected layer of 10 perceptrons instead of 1000 perceptrons.
- When our dataset is not similar to ImageNet dataset:

1. Freeze initial Convolution layers that extract low level features which are common for any Computer Vision problem (such as edges, basic shapes etc.) and fine-tune remaining Convolution layers
    - For example, freeze first 3 Convolution layers & fine-tune 4th and 5th Convolution layers.
2. Modify and fine-tune Fully Connected layers for our dataset

Since we are freezing many Convolution layers, no of parameters to be trained is significantly reduced. Hence, fine-tuning can be done faster in comparison to training from scratch.

---

14. VGG16 Model

1. References

Very Deep Convolutional Networks for Large-Scale Image Recognition Paper by Visual Geometry Group at University of Oxford

2. Intro

VGG16 is a CNN (Convolutional Neural Network) architecture which was used to win ILSVRC (Imagenet) competition in 2014. It was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is considered to be one of the excellent vision model architecture till date. The pre-trained version of the VGG16 network is trained on over 1 million images from the ImageNet visual database, and is able to classify images into 1,000 different categories with 92.7 % top-5 test accuracy.

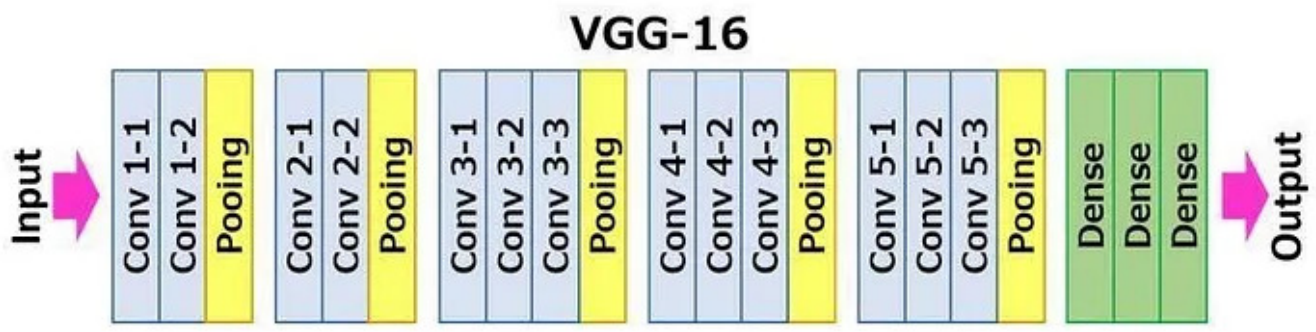
Note

Top-5 accuracy measures the percentage of test images for which the correct label is among the top five predicted labels out of 1000 labels of ImageNet challenge dataset.

3. VGG16 Architecture

In [ ]:
```
# VGG16 Layers

from IPython import display
display.Image("data/images/CV_Project_01_Emotion_Classifier_Keras-01-VGG16-Layers.jpg")
```

Out[ ]:

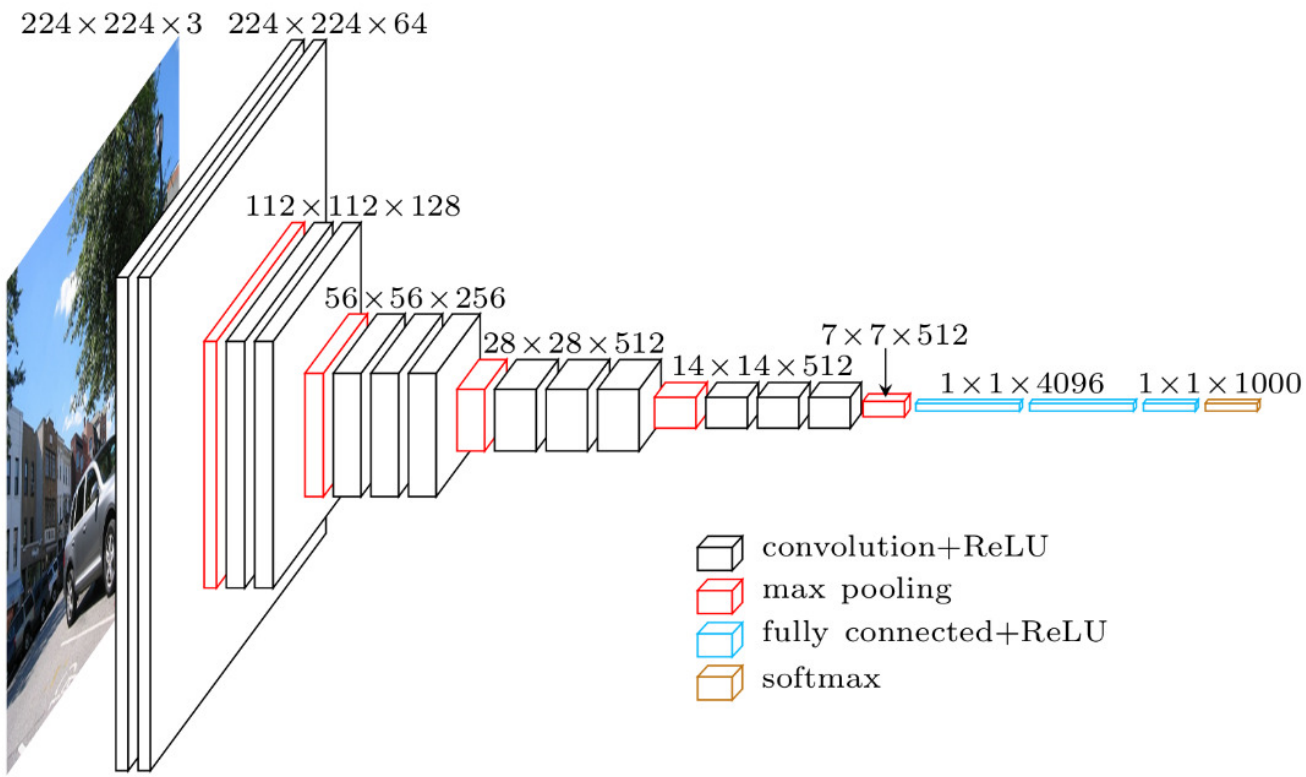


In the above figure, we have

- Convolution Layers
  - Conv 1 has 2 layers with 64 filters each
  - Conv 2 has 2 layers with 128 filters each
  - Conv 3 has 3 layers with 256 filters each
  - Conv 4 has 3 layers with 512 filters each
  - Conv 5 has 3 layers with 512 filters each
  - Total $2 + 2 + 3 + 3 + 3 = 13$ Convolution layers $\implies$ Responsible for feature extraction
  - All Convolution layers use 3×3 filter with stride 1 and `padding='same'` (no change in resolution before Pooling operation)
  - Maxpool layer of 2×2 with stride 2
- FC (Fully Connected) or Dense layers
  - 3 Fully Connected layers $\implies$ Responsible for classification
  - FC 1 has 4096 Perceptrons
  - FC 2 has 4096 Perceptrons
  - FC 3 has 1000 Perceptrons (since 1000 classes in ImageNet dataset)
- Total $13 + 3 = 16$ trainable layers $\implies$ model named as VGG16
- Final layer $\implies$ Softmax layer

In [ ]:
```
# VGG16 Architecture

from IPython import display
display.Image("data/images/CV_Project_01_Emotion_Classifier_Keras-02-VGG16-Architecture.
```

Out[ ]:

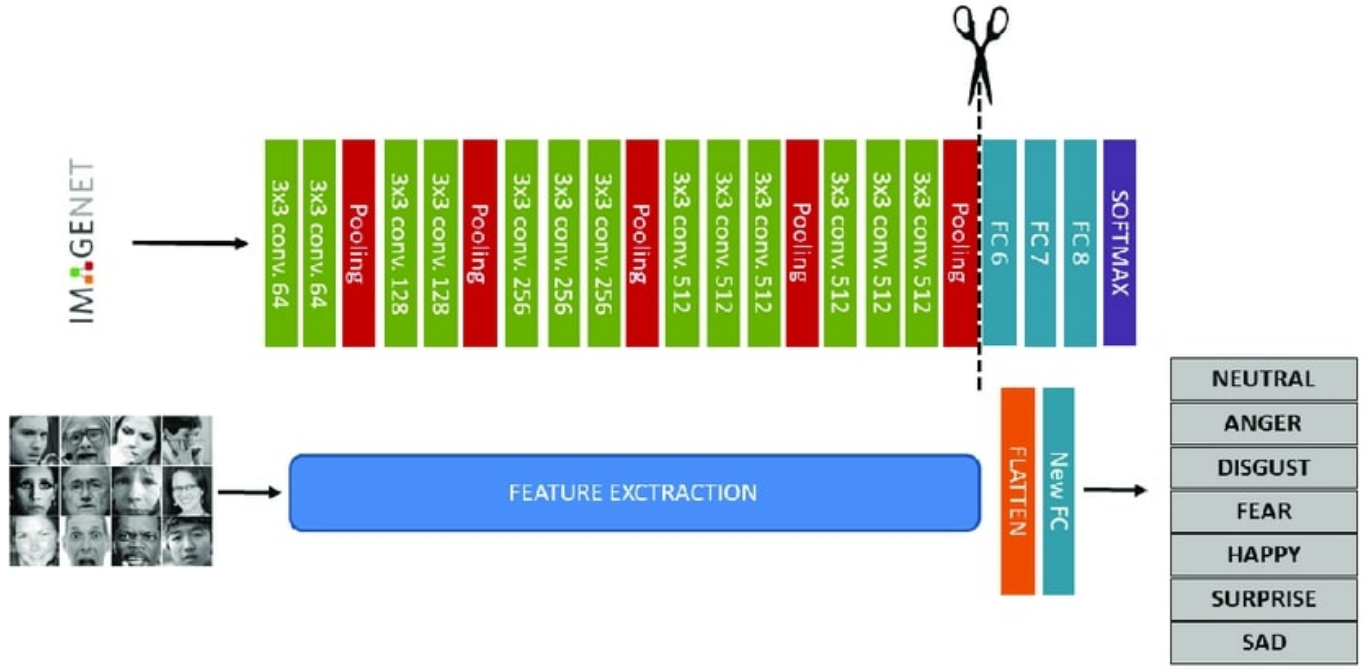


15. Model 3 : Transfer Learning VGG16

In [ ]:
```
# Transfer Learning VGG16 on FER-2013 Dataset
```

```
from IPython import display
display.Image("data/images/CV_Project_01_Emotion_Classifier_Keras-03-VGG16-Transfer-Lear
```

Out[ ]:



1. Data Loading: Load Images using `Keras ImageDataGenerator` with Data Augmentation

In [ ]:

```
# Define paths to the train and validation directories
train_data_dir = '/content/train'
test_data_dir = '/content/test'

# Set some parameters
img_width, img_height = 224, 224  # VGG16 is trained on 224 x 224 x 3 images
batch_size = 64
epochs = 10
num_classes = 7  # Update this based on the number of your classes

# Initializing the ImageDataGenerator with data augmentation options for the training se
data_generator = ImageDataGenerator(
    rescale=1./255,  # Rescale the pixel values from [0, 255] to [0, 1]
    rotation_range=10,  # Degree range for random rotations
    width_shift_range=0.1,  # Range (as a fraction of total width) for random horizontal
    height_shift_range=0.1,  # Range (as a fraction of total height) for random vertical
    zoom_range=0.2,  # Range for random zoom
    horizontal_flip=True,  # Randomly flip inputs horizontally
    fill_mode='nearest',  # Strategy to fill newly created pixels, which can appear afte
    validation_split=0.2  # Set the validation split; 20% of the data will be used for v
)

# No need for Data Augmentation for test data
test_preprocessor = ImageDataGenerator(
    rescale = 1./255,
)

# Automatically retrieve images and their classes for train, validation & test sets
train_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
```

```
    color_mode='rgb',
    subset='training',
    shuffle = True)

validation_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
    subset='validation')

test_generator = test_preprocessor.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb')
```

```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

2. Introduction of Class Weights to take care of Dataset Imbalance

Since our dataset is an imbalanced dataset, we need to provide a higher penalty for least represented classes in comparison to a lower penalty for classes with more representation in the dataset. This is achieved by Class Weights. We can use `compute_class_weight` function from `Scikit-Learn` to achieve this.

In [ ]:
```
# Extract class labels for all instances in the training dataset
classes = np.array(train_generator.classes)

# Calculate class weights to handle imbalances in the training data
# 'balanced' mode automatically adjusts weights inversely proportional to class frequenc
class_weights = compute_class_weight(
    class_weight='balanced',  # Strategy to balance classes
    classes=np.unique(classes),  # Unique class labels
    y=classes  # Class labels for each instance in the training dataset
)

# Create a dictionary mapping class indices to their calculated weights
class_weights_dict = dict(enumerate(class_weights))

# Print the class indices
print(f"Class Indices: \n{train_generator.class_indices}")

# Print the class weights dictionary
print(f"Class Weights Dictionary: \n{class_weights_dict}")
```

```
Class Indices:
{'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4, 'sad': 5, 'surprise': 6}
Class Weights Dictionary:
{0: 1.0266404434114071, 1: 9.401555464592715, 2: 1.0009587727708533, 3: 0.5684585684585685, 4: 0.826068191627104, 5: 0.8491570541259982, 6: 1.2933160650937552}
```

In our training set, we have 436 images for `disgust` class and 7215 images for `happy` class. Class weight for `disgust` class is 9.4066 while for `happy` class is 0.5684.

We have $\frac{7215}{436} = 16.55 = \frac{9.4066}{0.5684}$

3. Building Model

We are going to build the model by applying Transfer Learning on VGG16 trained on ImageNet dataset. We want only the convolution layers and we add our own dense layers on top of these convolution layers.

In [ ]:
```
# Clear the previous TensorFlow sessionx
tf.keras.backend.clear_session()

# Load the VGG16 base model, excluding its top (fully connected or dense) layers
vgg = VGG16(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

As you can see, all the parameters in above VGG16 model are trainable. We want to freeze the first 4 convolution blocks in VGG16 i.e. keep the parameters unchanged. We want to fine-tune the 5th convolution

block i.e. we want to make only those parameters trainable.

In [ ]:
```
# Make the specified layers non-trainable
for layer in vgg.layers[:-4]:
    layer.trainable = False

vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 7079424 (27.01 MB)
Non-trainable params: 7635264 (29.13 MB)
_________________________________________________________________
```

Now, we have only $2359808 \times 3 = 7079424$ trainable parameters corresponding to the 3 convolution layers in the 5th convolution block.

Now, we can add our own fully connected (dense) layers for Classification.

In [ ]:
```
# Flattening the layer and adding custom Dense layers
x = Flatten()(vgg.output)
```

```
# Adding a fully connected layer with ReLU activation and he_normal initializer
x = Dense(1024, activation='relu', kernel_initializer='he_normal')(x)
x = Dropout(0.5)(x)  # Adding dropout for regularization

# Adding another fully connected layer with ReLU activation and he_normal initializer
x = Dense(512, activation='relu', kernel_initializer='he_normal')(x)
x = Dropout(0.5)(x)  # Adding dropout for regularization

# Adding the output layer with softmax activation
# Note: Adjust the number of units to match the number of classes you have
output = Dense(num_classes, activation='softmax', kernel_initializer='he_normal')(x)

# Creating the final model
model = Model(inputs=vgg.input, outputs=output)

# Print the model summary
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 1024)              25691136

 dropout (Dropout)           (None, 1024)              0
```

```
 dense_1 (Dense)             (None, 512)               524800

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 7)                 3591

=================================================================
Total params: 40934215 (156.15 MB)
Trainable params: 33298951 (127.03 MB)
Non-trainable params: 7635264 (29.13 MB)
_________________________________________________________________
```

4. Callbacks

In [ ]:
```python
# File path for the model checkpoint
cnn_path = '/content/FER_2013_Emotion_Classifier/VGG16_Transfer_Learning'
name = 'VGG16_Transfer_Learning.keras'
chk_path = os.path.join(cnn_path, name)

# Callback to save the model checkpoint
checkpoint = ModelCheckpoint(filepath=chk_path,
                             save_best_only=True,
                             verbose=1,
                             mode='min',
                             monitor='val_loss')

# Callback for early stopping
earlystop = EarlyStopping(monitor='val_loss',
                          min_delta=0,
                          patience=3,
                          verbose=1,
                          restore_best_weights=True)

# Callback to reduce learning rate
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=6,
                              verbose=1,
                              min_delta=0.0001)

# Callback to log training data to a CSV file
csv_logger = CSVLogger(os.path.join(cnn_path,'training.log'))

# Aggregating all callbacks into a list
callbacks = [checkpoint, earlystop, reduce_lr, csv_logger]  # Adjusted as per your use-c
```

5. Training Model

In [ ]:
```python
# Compile the model
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad=Fa
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [ ]:
```python
# Calculate steps per epoch (no of batches needed to finish 1 epoch)
train_steps_per_epoch = np.ceil(train_generator.samples / train_generator.batch_size)
validation_steps_per_epoch = np.ceil(validation_generator.samples / validation_generator
test_steps_per_epoch = np.ceil(test_generator.samples / test_generator.batch_size)
print(f"train_steps_per_epoch = {train_steps_per_epoch}")
```

```
print(f"validation_steps_per_epoch = {validation_steps_per_epoch}")
print(f"test_steps_per_epoch = {test_steps_per_epoch}")
```

```
train_steps_per_epoch = 359.0
validation_steps_per_epoch = 90.0
test_steps_per_epoch = 113.0
```

In [ ]:

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_steps_per_epoch,
    epochs=100,
    validation_data=validation_generator,
    validation_steps=validation_steps_per_epoch,
    class_weight=class_weights_dict,
    callbacks=callbacks)
```

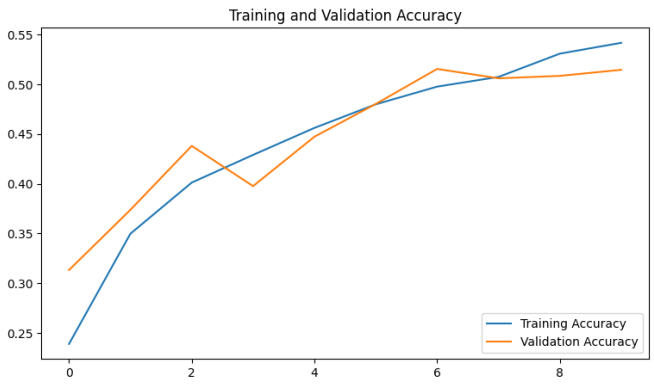
```
Epoch 1/100
359/359 [==============================] - ETA: 0s - loss: 1.8873 - accuracy: 0.2391
Epoch 1: val_loss improved from inf to 1.70381, saving model to /content/FER_2013_Emotio
n_Classifier/VGG16_Transfer_Learning/VGG16_Transfer_Learning.keras
359/359 [==============================] - 418s 1s/step - loss: 1.8873 - accuracy: 0.239
1 - val_loss: 1.7038 - val_accuracy: 0.3134 - lr: 1.0000e-04
Epoch 2/100
359/359 [==============================] - ETA: 0s - loss: 1.6785 - accuracy: 0.3498
Epoch 2: val_loss improved from 1.70381 to 1.61213, saving model to /content/FER_2013_Em
otion_Classifier/VGG16_Transfer_Learning/VGG16_Transfer_Learning.keras
359/359 [==============================] - 398s 1s/step - loss: 1.6785 - accuracy: 0.349
8 - val_loss: 1.6121 - val_accuracy: 0.3738 - lr: 1.0000e-04
Epoch 3/100
359/359 [==============================] - ETA: 0s - loss: 1.5469 - accuracy: 0.4012
Epoch 3: val_loss improved from 1.61213 to 1.44720, saving model to /content/FER_2013_Em
otion_Classifier/VGG16_Transfer_Learning/VGG16_Transfer_Learning.keras
359/359 [==============================] - 393s 1s/step - loss: 1.5469 - accuracy: 0.401
2 - val_loss: 1.4472 - val_accuracy: 0.4381 - lr: 1.0000e-04
Epoch 4/100
359/359 [==============================] - ETA: 0s - loss: 1.4716 - accuracy: 0.4290
Epoch 4: val_loss did not improve from 1.44720
359/359 [==============================] - 389s 1s/step - loss: 1.4716 - accuracy: 0.429
0 - val_loss: 1.5208 - val_accuracy: 0.3977 - lr: 1.0000e-04
Epoch 5/100
359/359 [==============================] - ETA: 0s - loss: 1.3926 - accuracy: 0.4562
Epoch 5: val_loss improved from 1.44720 to 1.41779, saving model to /content/FER_2013_Em
otion_Classifier/VGG16_Transfer_Learning/VGG16_Transfer_Learning.keras
359/359 [==============================] - 394s 1s/step - loss: 1.3926 - accuracy: 0.456
2 - val_loss: 1.4178 - val_accuracy: 0.4473 - lr: 1.0000e-04
Epoch 6/100
359/359 [==============================] - ETA: 0s - loss: 1.3280 - accuracy: 0.4799
Epoch 6: val_loss improved from 1.41779 to 1.34880, saving model to /content/FER_2013_Em
otion_Classifier/VGG16_Transfer_Learning/VGG16_Transfer_Learning.keras
359/359 [==============================] - 389s 1s/step - loss: 1.3280 - accuracy: 0.479
9 - val_loss: 1.3488 - val_accuracy: 0.4802 - lr: 1.0000e-04
Epoch 7/100
359/359 [==============================] - ETA: 0s - loss: 1.2795 - accuracy: 0.4976
Epoch 7: val_loss improved from 1.34880 to 1.25871, saving model to /content/FER_2013_Em
otion_Classifier/VGG16_Transfer_Learning/VGG16_Transfer_Learning.keras
359/359 [==============================] - 394s 1s/step - loss: 1.2795 - accuracy: 0.497
6 - val_loss: 1.2587 - val_accuracy: 0.5154 - lr: 1.0000e-04
Epoch 8/100
359/359 [==============================] - ETA: 0s - loss: 1.2405 - accuracy: 0.5074
Epoch 8: val_loss did not improve from 1.25871
359/359 [==============================] - 388s 1s/step - loss: 1.2405 - accuracy: 0.507
4 - val_loss: 1.2751 - val_accuracy: 0.5060 - lr: 1.0000e-04
Epoch 9/100
359/359 [==============================] - ETA: 0s - loss: 1.1737 - accuracy: 0.5308
Epoch 9: val_loss did not improve from 1.25871
359/359 [==============================] - 375s 1s/step - loss: 1.1737 - accuracy: 0.530
```

```
8 - val_loss: 1.3009 - val_accuracy: 0.5084 - lr: 1.0000e-04
Epoch 10/100
359/359 [==============================] - ETA: 0s - loss: 1.1563 - accuracy: 0.5417
Epoch 10: val_loss did not improve from 1.25871
Restoring model weights from the end of the best epoch: 7.
359/359 [==============================] - 377s 1s/step - loss: 1.1563 - accuracy: 0.541
7 - val_loss: 1.2609 - val_accuracy: 0.5145 - lr: 1.0000e-04
Epoch 10: early stopping
```

6. Plotting Performance Metrics

In [ ]:
```python
def plot_training_history(history, save_path=None):
    """
    Plots the training and validation accuracy and loss.

    Parameters:
    - history: A Keras History object. Contains the logs from the training process.

    Returns:
    - None. Displays the matplotlib plots for training/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 = range(len(acc))

    plt.figure(figsize=(20, 5))

    # Plot training and validation accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    # Plot training and validation loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')

    if save_path:
        plt.savefig(save_path)

    plt.show()
```

In [ ]:
```python
training_history_save_path = '/content/FER_2013_Emotion_Classifier/VGG16_Transfer_Learni
plot_training_history(history, training_history_save_path)
```



7. Model Evaluation

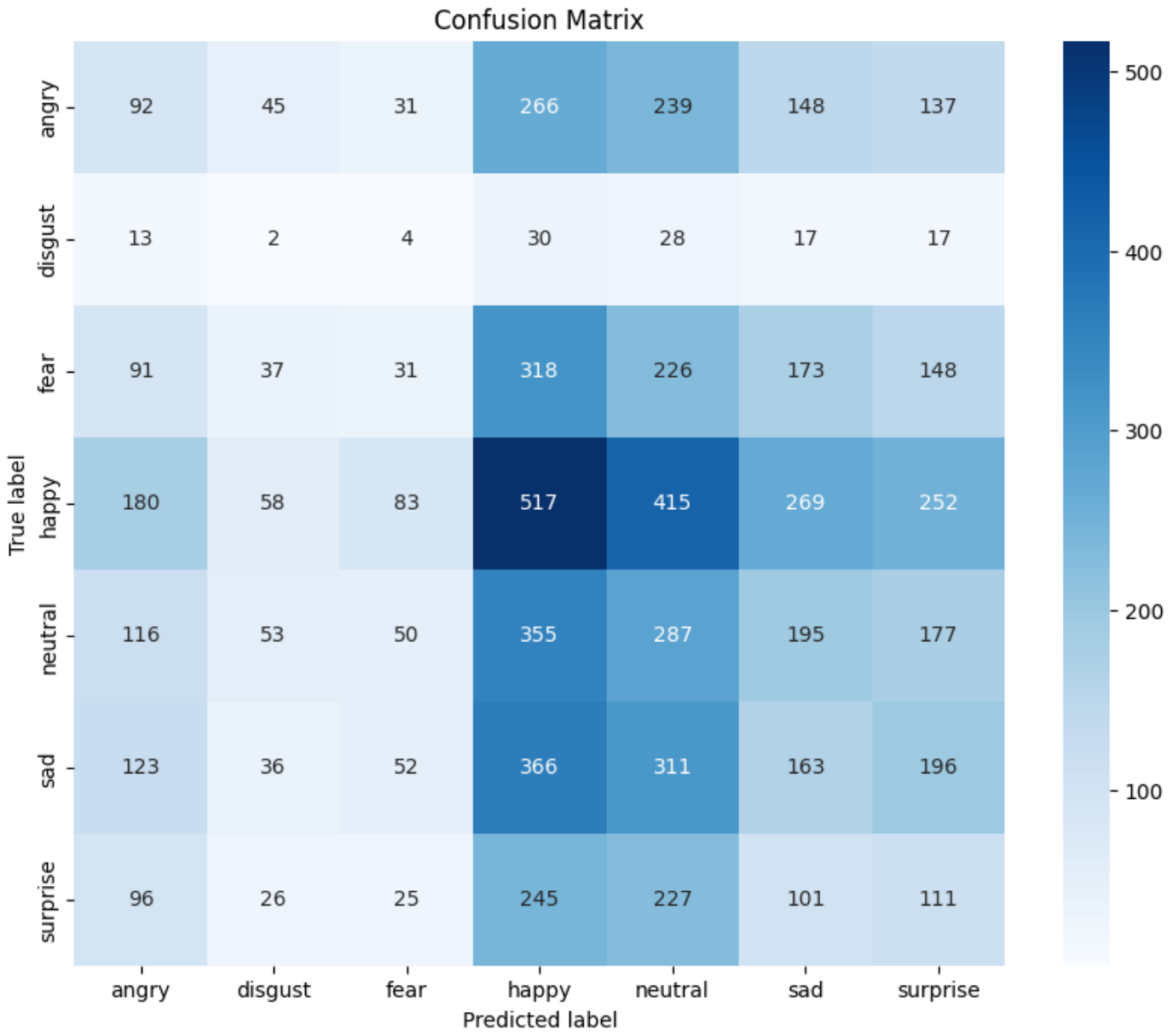
In [ ]:
```
train_loss, train_accu = model.evaluate(train_generator)
test_loss, test_accu = model.evaluate(test_generator)
print(f"Train accuracy = {train_accu*100:.2f} , Test accuracy = {test_accu*100:.2f}")
```

```
359/359 [==============================] - 302s 841ms/step - loss: 1.1924 - accuracy: 0.
5442
113/113 [==============================] - 33s 290ms/step - loss: 1.1923 - accuracy: 0.5
385
Train accuracy = 54.42 , Test accuracy = 53.85
```

8. Confusion Matrix

In [ ]:
```
# Assuming your true_classes and predicted_classes are already defined
true_classes = test_generator.classes
predicted_classes = np.argmax(model.predict(test_generator, steps=np.ceil(test_generator
class_labels = list(test_generator.class_indices.keys())

# Generate the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Plotting with seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')

confusion_matrix_save_path = '/content/FER_2013_Emotion_Classifier/VGG16_Transfer_Learni
plt.savefig(confusion_matrix_save_path)

plt.show()
```

```
113/113 [==============================] - 31s 272ms/step
```



9. Classification Report

In [ ]:

```
def save_classification_report(true_classes, predicted_classes, class_labels, save_path)
    """
    Generates a classification report and saves it as a PNG image.

    Parameters:
    - true_classes: Ground truth (correct) target values.
    - predicted_classes: Estimated targets as returned by a classifier.
    - class_labels: List of labels to index the matrix.
    - save_path: The path where the image will be saved.

    Returns:
    - None. Saves the classification report as a PNG image at the specified path.
    """
    report = classification_report(true_classes, predicted_classes, target_names=class_l

    # Create a matplotlib figure
    plt.figure(figsize=(10, 6))
    plt.text(0.01, 0.05, str(report), {'fontsize': 10}, fontproperties='monospace')
    plt.axis('off')

    # Save the figure
    plt.savefig(save_path, bbox_inches='tight', pad_inches=0.1)
```

```
    return report
```

In [ ]:
```
# Get Classification Report
classification_report_save_path = '/content/FER_2013_Emotion_Classifier/VGG16_Transfer_L
report = save_classification_report(true_classes, predicted_classes, class_labels, class
print(f"Classification Report:\n{report}")
```

```
Classification Report:
              precision    recall  f1-score   support

       angry       0.13      0.10      0.11       958
     disgust       0.01      0.02      0.01       111
        fear       0.11      0.03      0.05      1024
       happy       0.25      0.29      0.27      1774
     neutral       0.17      0.23      0.19      1233
         sad       0.15      0.13      0.14      1247
    surprise       0.11      0.13      0.12       831

    accuracy                           0.17      7178
   macro avg       0.13      0.13      0.13      7178
weighted avg       0.16      0.17      0.16      7178
```


```
              precision    recall  f1-score   support

       angry       0.13      0.10      0.11       958
     disgust       0.01      0.02      0.01       111
        fear       0.11      0.03      0.05      1024
       happy       0.25      0.29      0.27      1774
     neutral       0.17      0.23      0.19      1233
         sad       0.15      0.13      0.14      1247
    surprise       0.11      0.13      0.12       831

    accuracy                           0.17      7178
   macro avg       0.13      0.13      0.13      7178
weighted avg       0.16      0.17      0.16      7178
```


10. Making Predictions

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

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

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_generator) - 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_generator[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_generator[Random_batch][1][Random_Img_Index[i]], a

    # Making a prediction using the model
    Model_Prediction = np.argmax(model.predict(tf.expand_dims(Random_Img, axis=0), verbo

    # 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_Predic
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classe

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

11. Conclusion

Even though the acccuracy is good enough (53.85 % on test data), we can see many misclassifications in the random test batch above. If the model is good, we will have very high values across diagonal cells of the test data confusion matrix in comparison to off-diagonal cells. Upon inspecting the Confusion Matrix, we can clearly see the discrepancy. We can also see the poor values of Precision, Recall and F1-Score from the Classification Report. Hence from all these observations, we can draw the conclusion that our model is not a good one. Even Transfer Learning with VGG16 is not yielding good results.

---

16. ResNet50 Model

1. References

Deep Residual Learning for Image Recognition Paper by Microsoft Research

2. ResNet Model: Intro

Every consecutive winning architecture uses more layers in a deep neural network to lower the error rate after the first CNN-based architecture (AlexNet) that won the ImageNet 2012 competition. This is effective for smaller numbers of layers, but when we add more layers, a typical deep learning issue known as the **Vanishing or Exploding gradient** arises. This results in the gradient becoming zero or being overly large. Therefore, the training and test error rate similarly increases as the number of layers is increased.

Vanishing and Exploding gradients are explained below:

1. Vanishing Gradients
    - During backpropagation, the error signal (gradient) gets super small as it travels back through the layers. By the time it reaches the earlier layers, the gradient becomes negligible. Thus hardly any change comes to weights, making it hard to learn.
    - This is common with activation functions like sigmoid that flatten out at the edges. The function doesn't change much for large inputs. So, the gradient for those large errors becomes tiny when multiplied back through the layers.
2. Exploding Gradients
    - This is the opposite. The error signal (gradient) explodes as it travels back. By the time it reaches the earlier layers, the gradient is too large and the adjustments become huge and unstable, messing up the whole learning process.
    - This can happen if weights are not properly initialized or with some activation functions. Like a snowball rolling down a hill, the gradient keeps getting bigger with each multiplication.

In both cases, the network struggles to learn effectively. Vanishing gradients make the early layers unresponsive, while exploding gradients make everything unstable.

We can see from the following figure that a 20-layer CNN architecture performs better on training and testing datasets than a 56-layer CNN architecture.

In [ ]:
```
# Training & Test errors on CIFAR-10 for 20 and 56 layer plain networks

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-01.jpg")
```

Out[ ]:

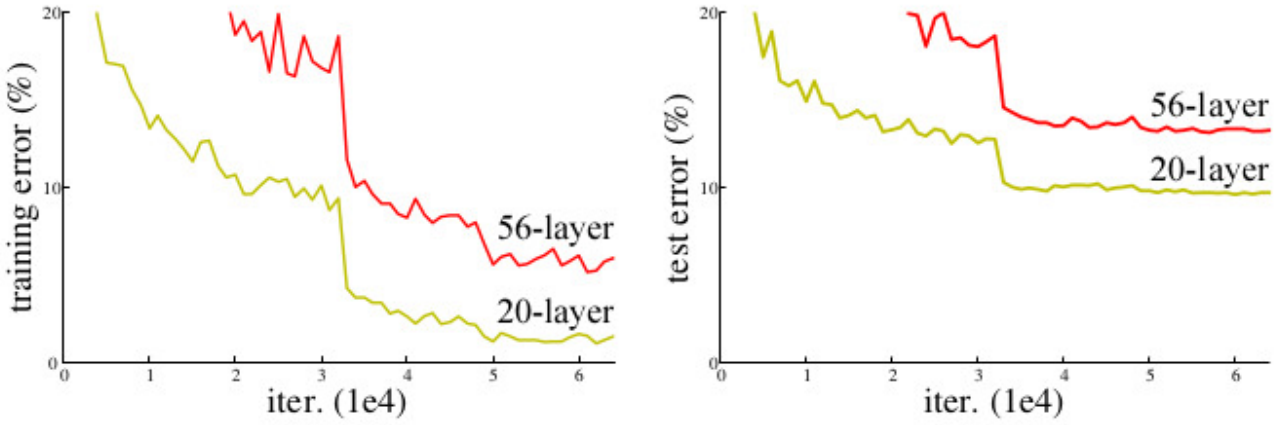


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

After lots of experimentation with different datasets (like CIFAR-10 and ImageNet) on plain networks (like VGG16, VGG19 etc), researchers at Microsoft Research arrived at the below conclusion:

- For a plain network, even if we increase the number of layers, we cannot assume that the model will perform better.
- The error rate is caused by a vanishing / exploding gradient after further analysis of the error rate.

In order to solve this problem, they have proposed a technique called **Residual Learning**.

3. Residual Learning

In [ ]:
```
# Residual / Identity / Skip Connection - Building block of Residual Learning

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-02.jpg")
```

Out[ ]:

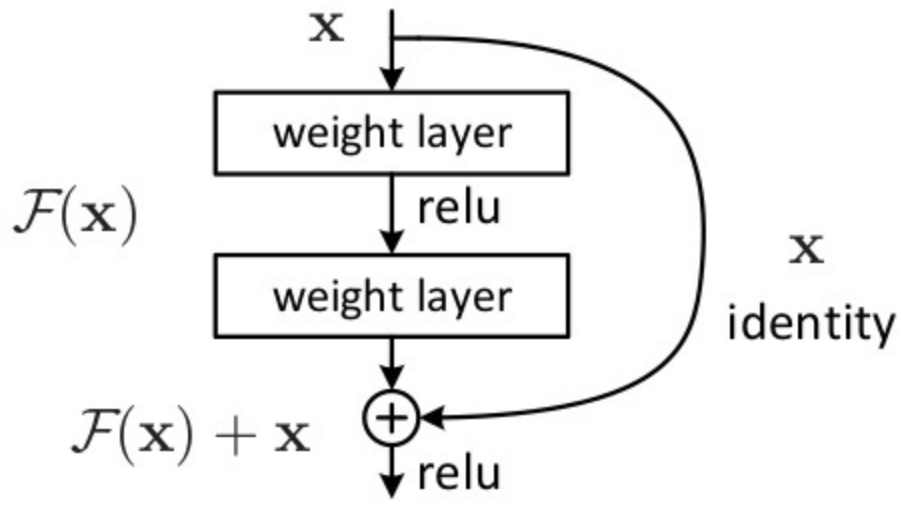


Figure 2. Residual learning: a building block.

1. Residual / Identity / Skip Connections:

   In order to address the issue of the Vanishing / Exploding gradients, ResNets (Residual Networks) use Residual / Identity / Skip connections as building blocks. In these Skip connections, we add the original input to the output of the convolutional block.

   Skip connection is a direct connection that skips over some layers of the model.

   Without the skip connection, we have $Y = F(w \cdot x + b) = F(X)$.

   With skip connection, we have $Y = F(X) + X$.

   Note 1

   In order to understand ResNets, we can use an analogy. Let individual layers be students in a class. As you stack more layers (i.e. adding more students), training gets harder for the earlier ones because of Vanishing / Exploding gradients. ResNets add a shortcut connection or skip connection, like a cheat sheet, to help these layers (students) learn.

   Imagine a student (layer) has trouble understanding a complex concept. A ResNet allows the student to directly access the original information (from a previous layer) along with what they've learned so far. This helps them learn the difference (residual) between the original and the transformed information.

   By using skip connections, ResNets can train much deeper networks and avoid the Vanishing / Exploding gradient problems, making them powerful tools for image recognition and other tasks.

   Note 2

   Another way to explain Skip connections is as follows. If all weights in one convolution layer are negative, ReLU activation makes all of them 0's which are passed to next layer. Hence, network does not learn anything.

   When we use Skip connections, even if weights are all negative, same input to that layer is added later. Hence, network can learn. If weights are positive, then we are adding extra information. Here also, network can learn. Thus, we can solve the Vanishing / Exploding gradient problems and train much deeper networks.

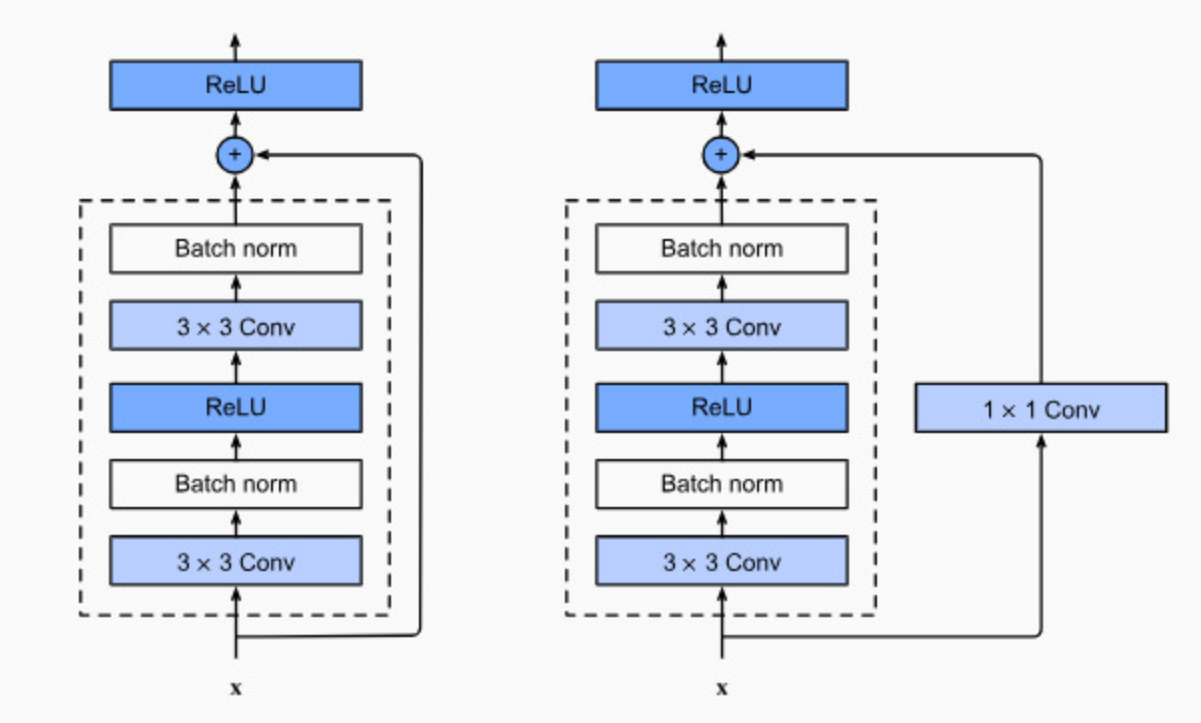
2. Conditions for Skip Connection: The skip connection works because it adds information with the same dimensions. There are two main cases:

   - Same input and output size:
     - This is the most common case. The original information and the transformed information have the same dimensions, so adding them directly makes sense. Most residual blocks in ResNets use this approach.
     - Example $\implies$ Identity Block
   - Downsampling:
     - In some cases, the network might reduce the size of the information as it goes through layers. Here, the skip connection might use techniques like
       - padding with zeros to match the dimensions before adding
       - performing 1x1 convolutions

- Example $\Longrightarrow$ Convolutional Block: In a Convolutional Block, a 1x1 convolution is applied to the input data to adjust its dimensions before it can be added to the output of the convolutional layers. This is because the input and output data have different dimensions.

In [ ]:
```
# Identity Block and Convolutional Block

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-02-02.jpg")
```

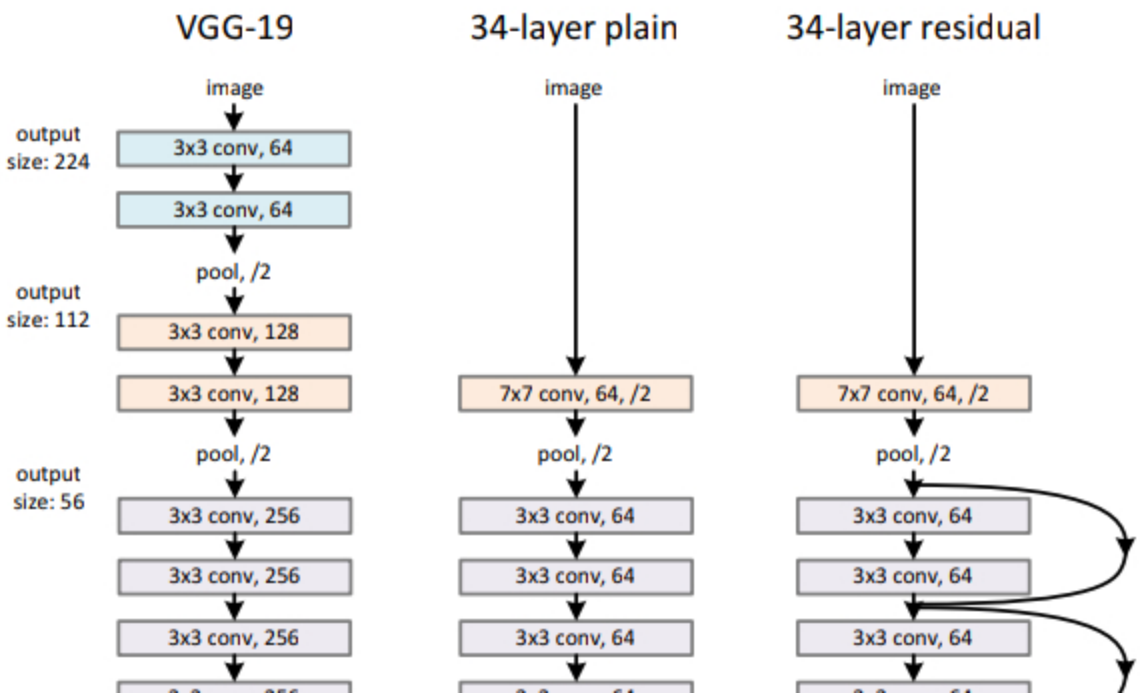
Out[ ]:



4. Comparison of Plain and Residual Networks

In [ ]:
```
# Comparison of Architectures of Plain and Residual Networks for ImageNet

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-03.jpg")
```

Out[ ]:



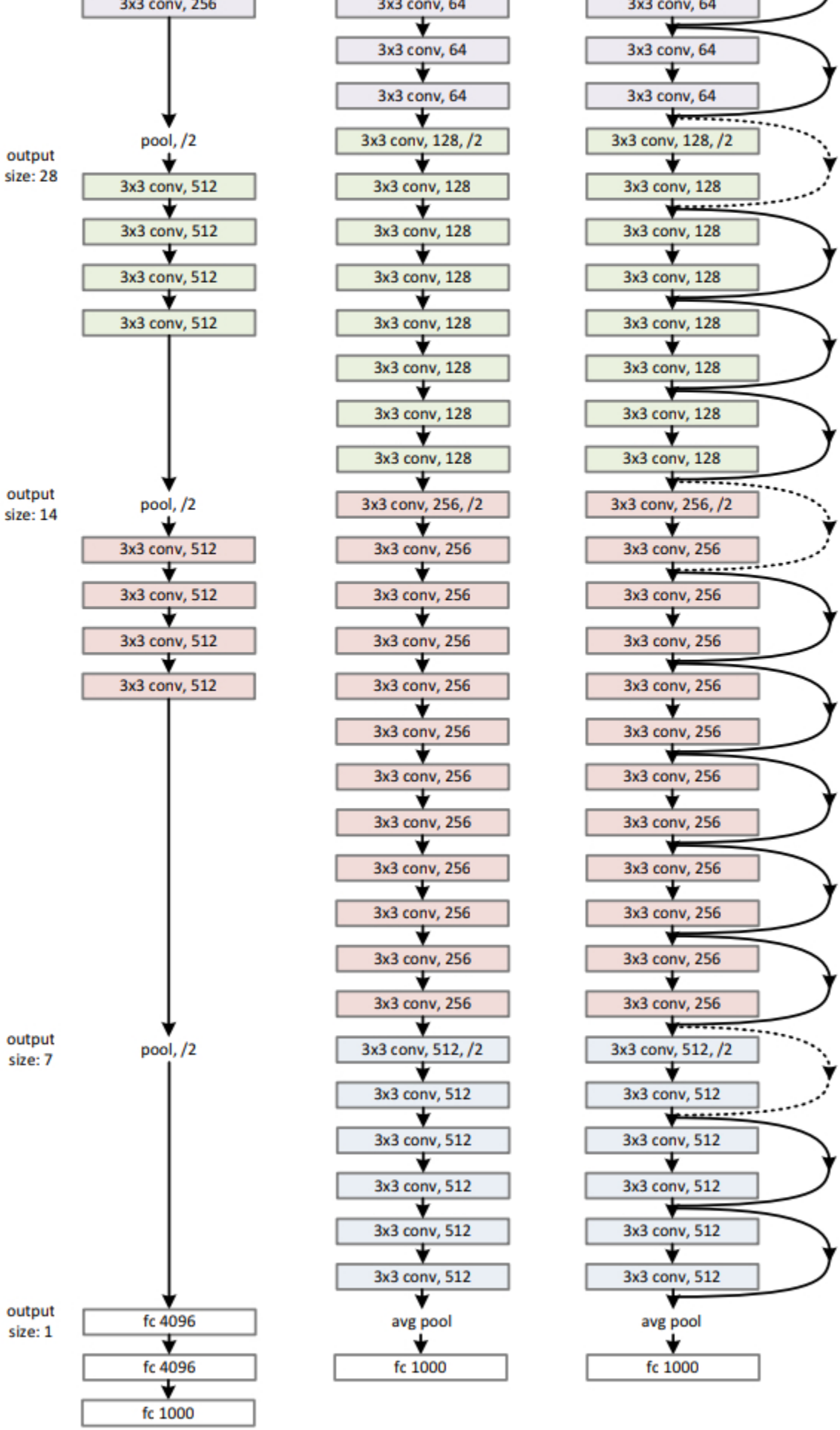


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [40] (19.6 billion FLOPs) as a reference. **Mid-**

dle: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

In [ ]:
```
# Training result curves of Plain and Residual Networks for ImageNet

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-04.jpg")
```

Out[ ]:

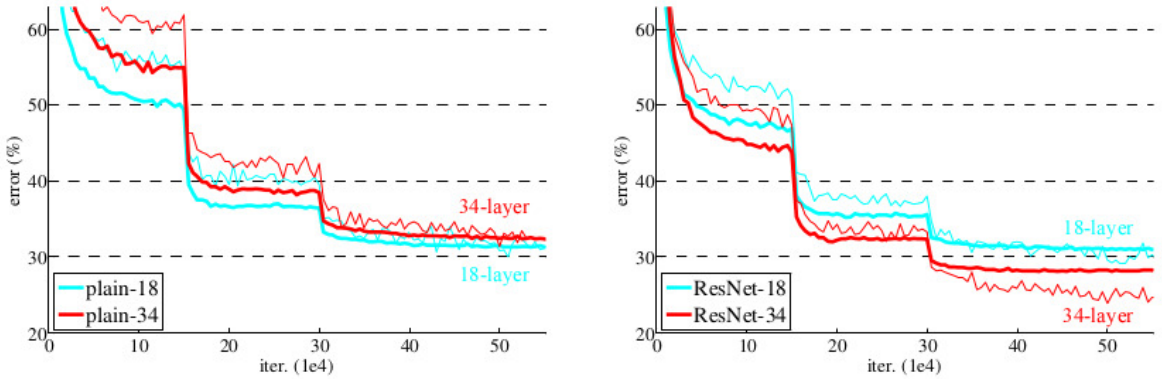


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

In [ ]:
```
# Training result table of Plain and Residual Networks for ImageNet

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-05.jpg")
```

Out[ ]:

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

5. Deeper Bottleneck Architectures

In [ ]:
```
# Residual Building blocks for ImageNet (2-layer and 3-layer)

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-06.jpg")
```

Out[ ]:

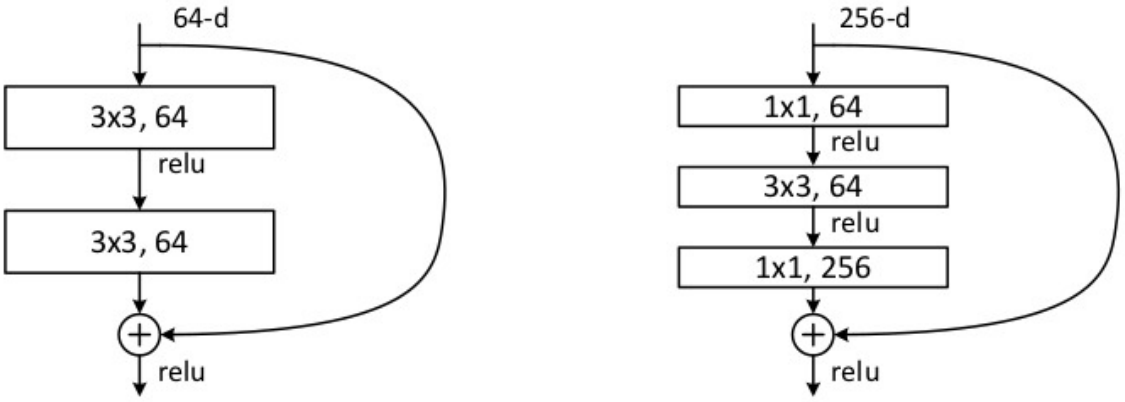


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

- ResNet-18 and ResNet-34 $\Longrightarrow$ Residual building block is a stack of 2 convolution layers each with 64 3x3 kernels
- ResNet-50, ResNet-101 and ResNet-152 $\Longrightarrow$ Residual building block is a stack of 3 convolution layers
  - 1st one with 64 1x1 kernels
  - 2nd one with 64 3x3 kernels
  - 3rd one with 256 1x1 kernels

6. ResNet Architectures for ImageNet

In [ ]:
```
# ResNet Architectures for ImageNet

from IPython import display
display.Image("data/images/CV_05_ResNet_Model-07.jpg")
```

Out[ ]:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

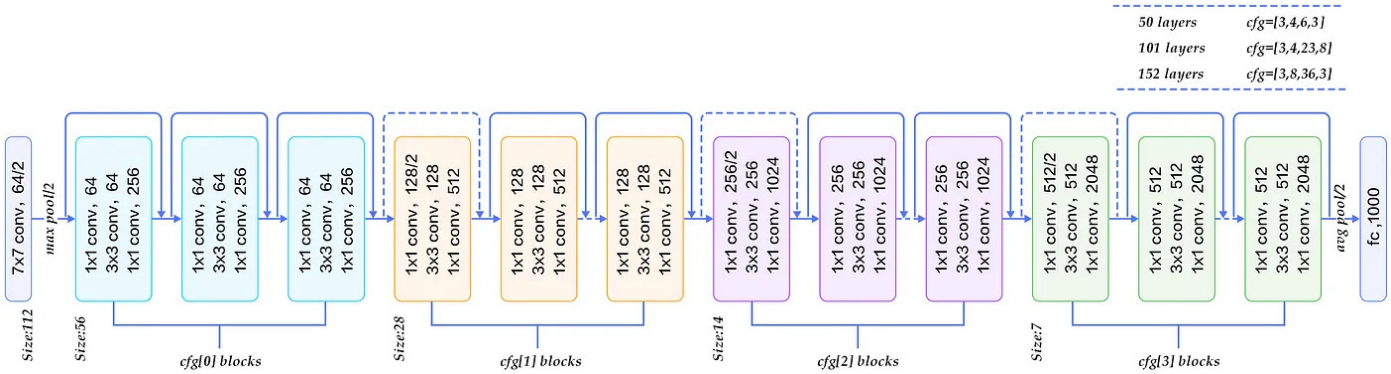
Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

7. ResNet50 Architecture

In [ ]:
```
# ResNet50 Architecture
```

```
from IPython import display
display.Image("data/images/CV_05_ResNet_Model-08-ResNet50-Architecture.jpg")
```

Out[ ]:



17. Model 4 : Transfer Learning ResNet50

1. Data Loading: Load Images using `Keras ImageDataGenerator` with Data Augmentation

In [ ]:

```
# Define paths to the train and validation directories
train_data_dir = '/content/train'
test_data_dir = '/content/test'

# Set some parameters
img_width, img_height = 224, 224  # ResNet50 is trained on 224 x 224 x 3 images
batch_size = 64
epochs = 20
num_classes = 7  # Update this based on the number of your classes

# Initializing the ImageDataGenerator with data augmentation options for the training se
data_generator = ImageDataGenerator(
    rescale=1./255,  # Rescale the pixel values from [0, 255] to [0, 1]
    rotation_range=10,  # Degree range for random rotations
    width_shift_range=0.1,  # Range (as a fraction of total width) for random horizontal
    height_shift_range=0.1,  # Range (as a fraction of total height) for random vertical
    zoom_range=0.2,  # Range for random zoom
    horizontal_flip=True,  # Randomly flip inputs horizontally
    fill_mode='nearest',  # Strategy to fill newly created pixels, which can appear afte
    validation_split=0.2  # Set the validation split; 20% of the data will be used for v
)

# No need for Data Augmentation for test data
test_preprocessor = ImageDataGenerator(
    rescale = 1./255,
)

# Automatically retrieve images and their classes for train, validation & test sets
train_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
    subset='training',
    shuffle = True)
```

```
validation_generator = data_generator.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
    subset='validation')

test_generator = test_preprocessor.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=False)
```

```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

2. Introduction of Class Weights to take care of Dataset Imbalance

Since our dataset is an imbalanced dataset, we need to provide a higher penalty for least represented classes in comparison to a lower penalty for classes with more representation in the dataset. This is achieved by Class Weights. We can use `compute_class_weight` function from `Scikit-Learn` to achieve this.

In [ ]:
```
# Extract class labels for all instances in the training dataset
classes = np.array(train_generator.classes)

# Calculate class weights to handle imbalances in the training data
# 'balanced' mode automatically adjusts weights inversely proportional to class frequenc
class_weights = compute_class_weight(
    class_weight='balanced',  # Strategy to balance classes
    classes=np.unique(classes),  # Unique class labels
    y=classes  # Class labels for each instance in the training dataset
)

# Create a dictionary mapping class indices to their calculated weights
class_weights_dict = dict(enumerate(class_weights))

# Print the class indices
print(f"Class Indices: \n{train_generator.class_indices}")

# Print the class weights dictionary
print(f"Class Weights Dictionary: \n{class_weights_dict}")
```

```
Class Indices:
{'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4, 'sad': 5, 'surprise': 6}
Class Weights Dictionary:
{0: 1.0266404434114071, 1: 9.401555464592715, 2: 1.0009587727708533, 3: 0.56845856845856
85, 4: 0.826068191627104, 5: 0.8491570541259982, 6: 1.2933160650937552}
```

In our training set, we have 436 images for `disgust` class and 7215 images for `happy` class. Class weight for `disgust` class is 9.4066 while for `happy` class is 0.5684.

We have $\frac{7215}{436} = 16.55 = \frac{9.4066}{0.5684}$

3. Building Model

We are going to build the model by applying Transfer Learning on ResNet50 trained on ImageNet dataset. We want only the convolution layers and we add our own dense layers on top of these convolution layers.

In [ ]:
```
# Clear the previous TensorFlow sessionx
tf.keras.backend.clear_session()

# Load the ResNet50V2 base model, excluding its top (fully connected or dense) layers
resnet = ResNet50V2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
resnet.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resne
t/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94668760/94668760 [==============================] - 0s 0us/step
Model: "resnet50v2"
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to

==========================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         []


 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)       0         ['input_1[0][0]']


 conv1_conv (Conv2D)         (None, 112, 112, 64)      9472      ['conv1_pad[0][0]']


 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)      0         ['conv1_conv[0][0]']


 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)        0         ['pool1_pad[0][0]']


 conv2_block1_preact_bn (Ba  (None, 56, 56, 64)        256       ['pool1_pool[0][0]']
 tchNormalization)


 conv2_block1_preact_relu (  (None, 56, 56, 64)        0         ['conv2_block1_preac
t_bn[0][0]
 Activation)                                                     ']


 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)        4096      ['conv2_block1_preac
t_relu[0][
 D)                                                              0']


 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)        256       ['conv2_block1_1_con
v[0][0]']
 rmalization)
```

```
 conv2_block1_1_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block1_1_bn
[0][0]']
 ation)


 conv2_block1_2_pad (ZeroPa  (None, 58, 58, 64)          0         ['conv2_block1_1_rel
u[0][0]']
 dding2D)


 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)          36864     ['conv2_block1_2_pad
[0][0]']
 D)


 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)          256       ['conv2_block1_2_con
v[0][0]']
 rmalization)


 conv2_block1_2_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block1_2_bn
[0][0]']
 ation)


 conv2_block1_0_conv (Conv2  (None, 56, 56, 256)         16640     ['conv2_block1_preac
t_relu[0][
 D)                                                      0]']


 conv2_block1_3_conv (Conv2  (None, 56, 56, 256)         16640     ['conv2_block1_2_rel
u[0][0]']
 D)


 conv2_block1_out (Add)      (None, 56, 56, 256)         0         ['conv2_block1_0_con
v[0][0]',
                                                          'conv2_block1_3_con
v[0][0]']

 conv2_block2_preact_bn (Ba  (None, 56, 56, 256)         1024      ['conv2_block1_out
[0][0]']
 tchNormalization)


 conv2_block2_preact_relu (  (None, 56, 56, 256)         0         ['conv2_block2_preac
t_bn[0][0]
 Activation)                                             ']


 conv2_block2_1_conv (Conv2  (None, 56, 56, 64)          16384     ['conv2_block2_preac
t_relu[0][
 D)                                                      0]']
```

```
 conv2_block2_1_bn (BatchNo  (None, 56, 56, 64)          256       ['conv2_block2_1_con
v[0][0]']
 rmalization)


 conv2_block2_1_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block2_1_bn
[0][0]']
 ation)


 conv2_block2_2_pad (ZeroPa  (None, 58, 58, 64)          0         ['conv2_block2_1_rel
u[0][0]']
 dding2D)


 conv2_block2_2_conv (Conv2  (None, 56, 56, 64)          36864     ['conv2_block2_2_pad
[0][0]']
 D)


 conv2_block2_2_bn (BatchNo  (None, 56, 56, 64)          256       ['conv2_block2_2_con
v[0][0]']
 rmalization)


 conv2_block2_2_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block2_2_bn
[0][0]']
 ation)


 conv2_block2_3_conv (Conv2  (None, 56, 56, 256)         16640     ['conv2_block2_2_rel
u[0][0]']
 D)


 conv2_block2_out (Add)      (None, 56, 56, 256)         0         ['conv2_block1_out
[0][0]',
                                                                  'conv2_block2_3_con
v[0][0]']

 conv2_block3_preact_bn (Ba  (None, 56, 56, 256)         1024      ['conv2_block2_out
[0][0]']
 tchNormalization)


 conv2_block3_preact_relu (  (None, 56, 56, 256)         0         ['conv2_block3_preac
t_bn[0][0]
 Activation)                                                      ']


 conv2_block3_1_conv (Conv2  (None, 56, 56, 64)          16384     ['conv2_block3_preac
t_relu[0][
 D)                                                               0]']
```

```
 conv2_block3_1_bn (BatchNo  (None, 56, 56, 64)          256       ['conv2_block3_1_con
v[0][0]']
 rmalization)


 conv2_block3_1_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block3_1_bn
[0][0]']
 ation)


 conv2_block3_2_pad (ZeroPa  (None, 58, 58, 64)          0         ['conv2_block3_1_rel
u[0][0]']
 dding2D)


 conv2_block3_2_conv (Conv2  (None, 28, 28, 64)          36864     ['conv2_block3_2_pad
[0][0]']
 D)


 conv2_block3_2_bn (BatchNo  (None, 28, 28, 64)          256       ['conv2_block3_2_con
v[0][0]']
 rmalization)


 conv2_block3_2_relu (Activ  (None, 28, 28, 64)          0         ['conv2_block3_2_bn
[0][0]']
 ation)


 max_pooling2d (MaxPooling2  (None, 28, 28, 256)         0         ['conv2_block2_out
[0][0]']
 D)


 conv2_block3_3_conv (Conv2  (None, 28, 28, 256)         16640     ['conv2_block3_2_rel
u[0][0]']
 D)


 conv2_block3_out (Add)      (None, 28, 28, 256)         0         ['max_pooling2d[0]
[0]',
                                                                  'conv2_block3_3_con
v[0][0]']

 conv3_block1_preact_bn (Ba  (None, 28, 28, 256)         1024      ['conv2_block3_out
[0][0]']
 tchNormalization)


 conv3_block1_preact_relu (  (None, 28, 28, 256)         0         ['conv3_block1_preac
t_bn[0][0]
 Activation)                                                      ']
```

```
 conv3_block1_1_conv (Conv2  (None, 28, 28, 128)          32768     ['conv3_block1_preac
 D)                                                                t_relu[0][
                                                                   0]']

 conv3_block1_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block1_1_con
 rmalization)                                                      v[0][0]']

 conv3_block1_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block1_1_bn
 ation)                                                            [0][0]']

 conv3_block1_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block1_1_rel
 dding2D)                                                          u[0][0]']

 conv3_block1_2_conv (Conv2  (None, 28, 28, 128)          147456    ['conv3_block1_2_pad
 D)                                                                [0][0]']

 conv3_block1_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block1_2_con
 rmalization)                                                      v[0][0]']

 conv3_block1_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block1_2_bn
 ation)                                                            [0][0]']

 conv3_block1_0_conv (Conv2  (None, 28, 28, 512)          131584    ['conv3_block1_preac
 D)                                                                t_relu[0][
                                                                   0]']

 conv3_block1_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block1_2_rel
 D)                                                                u[0][0]']

 conv3_block1_out (Add)      (None, 28, 28, 512)          0         ['conv3_block1_0_con
                                                                   v[0][0]',
                                                                    'conv3_block1_3_con
                                                                   v[0][0]']

 conv3_block2_preact_bn (Ba  (None, 28, 28, 512)          2048      ['conv3_block1_out
 tchNormalization)                                                 [0][0]']
```

```
 conv3_block2_preact_relu (  (None, 28, 28, 512)          0         ['conv3_block2_preac
t_bn[0][0]
 Activation)                                                        ']


 conv3_block2_1_conv (Conv2  (None, 28, 28, 128)          65536     ['conv3_block2_preac
t_relu[0][
 D)                                                                 0]']


 conv3_block2_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block2_1_con
v[0][0]']
 rmalization)


 conv3_block2_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block2_1_bn
[0][0]']
 ation)


 conv3_block2_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block2_1_rel
u[0][0]']
 dding2D)


 conv3_block2_2_conv (Conv2  (None, 28, 28, 128)          147456    ['conv3_block2_2_pad
[0][0]']
 D)


 conv3_block2_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block2_2_con
v[0][0]']
 rmalization)


 conv3_block2_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block2_2_bn
[0][0]']
 ation)


 conv3_block2_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block2_2_rel
u[0][0]']
 D)


 conv3_block2_out (Add)      (None, 28, 28, 512)          0         ['conv3_block1_out
[0][0]',
                                                                    'conv3_block2_3_con
v[0][0]']


 conv3_block3_preact_bn (Ba  (None, 28, 28, 512)          2048      ['conv3_block2_out
[0][0]']
 tchNormalization)
```

```
 conv3_block3_preact_relu (  (None, 28, 28, 512)          0         ['conv3_block3_preac
t_bn[0][0]
 Activation)                                                        ']


 conv3_block3_1_conv (Conv2  (None, 28, 28, 128)          65536     ['conv3_block3_preac
t_relu[0][
 D)                                                                 0]']


 conv3_block3_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block3_1_con
v[0][0]']
 rmalization)


 conv3_block3_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block3_1_bn
[0][0]']
 ation)


 conv3_block3_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block3_1_rel
u[0][0]']
 dding2D)


 conv3_block3_2_conv (Conv2  (None, 28, 28, 128)          147456    ['conv3_block3_2_pad
[0][0]']
 D)


 conv3_block3_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block3_2_con
v[0][0]']
 rmalization)


 conv3_block3_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block3_2_bn
[0][0]']
 ation)


 conv3_block3_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block3_2_rel
u[0][0]']
 D)


 conv3_block3_out (Add)      (None, 28, 28, 512)          0         ['conv3_block2_out
[0][0]',
                                                                    'conv3_block3_3_con
v[0][0]']

 conv3_block4_preact_bn (Ba  (None, 28, 28, 512)          2048      ['conv3_block3_out
[0][0]']
 tchNormalization)
```

```
 conv3_block4_preact_relu (  (None, 28, 28, 512)          0         ['conv3_block4_preac
t_bn[0][0]
 Activation)                                                        ']


 conv3_block4_1_conv (Conv2  (None, 28, 28, 128)          65536     ['conv3_block4_preac
t_relu[0][
 D)                                                                 0]']


 conv3_block4_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block4_1_con
v[0][0]']
 rmalization)


 conv3_block4_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block4_1_bn
[0][0]']
 ation)


 conv3_block4_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block4_1_rel
u[0][0]']
 dding2D)


 conv3_block4_2_conv (Conv2  (None, 14, 14, 128)          147456    ['conv3_block4_2_pad
[0][0]']
 D)


 conv3_block4_2_bn (BatchNo  (None, 14, 14, 128)          512       ['conv3_block4_2_con
v[0][0]']
 rmalization)


 conv3_block4_2_relu (Activ  (None, 14, 14, 128)          0         ['conv3_block4_2_bn
[0][0]']
 ation)


 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 512)          0         ['conv3_block3_out
[0][0]']
 g2D)


 conv3_block4_3_conv (Conv2  (None, 14, 14, 512)          66048     ['conv3_block4_2_rel
u[0][0]']
 D)


 conv3_block4_out (Add)      (None, 14, 14, 512)          0         ['max_pooling2d_1[0]
[0]',
                                                                     'conv3_block4_3_con
v[0][0]']
```

```
 conv4_block1_preact_bn (Ba  (None, 14, 14, 512)          2048      ['conv3_block4_out
[0][0]']
 tchNormalization)


 conv4_block1_preact_relu (  (None, 14, 14, 512)          0         ['conv4_block1_preac
t_bn[0][0]
 Activation)                                                        ']


 conv4_block1_1_conv (Conv2  (None, 14, 14, 256)          131072    ['conv4_block1_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block1_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block1_1_con
v[0][0]']
 rmalization)


 conv4_block1_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block1_1_bn
[0][0]']
 ation)


 conv4_block1_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block1_1_rel
u[0][0]']
 dding2D)


 conv4_block1_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block1_2_pad
[0][0]']
 D)


 conv4_block1_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block1_2_con
v[0][0]']
 rmalization)


 conv4_block1_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block1_2_bn
[0][0]']
 ation)


 conv4_block1_0_conv (Conv2  (None, 14, 14, 1024)         525312    ['conv4_block1_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block1_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block1_2_rel
u[0][0]']
 D)
```

```
 conv4_block1_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block1_0_con
v[0][0]',
                                                                    'conv4_block1_3_con
v[0][0]']

 conv4_block2_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block1_out
[0][0]']
 tchNormalization)

 conv4_block2_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block2_preac
t_bn[0][0]
 Activation)                                                        ']

 conv4_block2_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block2_preac
t_relu[0][
 D)                                                                 0]']

 conv4_block2_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block2_1_con
v[0][0]']
 rmalization)

 conv4_block2_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block2_1_bn
[0][0]']
 ation)

 conv4_block2_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block2_1_rel
u[0][0]']
 dding2D)

 conv4_block2_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block2_2_pad
[0][0]']
 D)

 conv4_block2_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block2_2_con
v[0][0]']
 rmalization)

 conv4_block2_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block2_2_bn
[0][0]']
 ation)

 conv4_block2_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block2_2_rel
u[0][0]']
 D)
```

```
 conv4_block2_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block1_out
[0][0]',
                                                                    'conv4_block2_3_con
v[0][0]']

 conv4_block3_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block2_out
[0][0]']
 tchNormalization)

 conv4_block3_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block3_preac
t_bn[0][0]
 Activation)                                                        ']

 conv4_block3_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block3_preac
t_relu[0][
 D)                                                                 0]']

 conv4_block3_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block3_1_con
v[0][0]']
 rmalization)

 conv4_block3_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block3_1_bn
[0][0]']
 ation)

 conv4_block3_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block3_1_rel
u[0][0]']
 dding2D)

 conv4_block3_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block3_2_pad
[0][0]']
 D)

 conv4_block3_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block3_2_con
v[0][0]']
 rmalization)

 conv4_block3_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block3_2_bn
[0][0]']
 ation)

 conv4_block3_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block3_2_rel
u[0][0]']
 D)
```

```
 conv4_block3_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block2_out
[0][0]',
                                                                    'conv4_block3_3_con
v[0][0]']

 conv4_block4_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block3_out
[0][0]']
 tchNormalization)

 conv4_block4_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block4_preac
t_bn[0][0]
 Activation)                                                        ']

 conv4_block4_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block4_preac
t_relu[0][
 D)                                                                 0]']

 conv4_block4_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block4_1_con
v[0][0]']
 rmalization)

 conv4_block4_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block4_1_bn
[0][0]']
 ation)

 conv4_block4_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block4_1_rel
u[0][0]']
 dding2D)

 conv4_block4_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block4_2_pad
[0][0]']
 D)

 conv4_block4_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block4_2_con
v[0][0]']
 rmalization)

 conv4_block4_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block4_2_bn
[0][0]']
 ation)

 conv4_block4_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block4_2_rel
u[0][0]']
 D)
```

```
 conv4_block4_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block3_out
[0][0]',
                                                                    'conv4_block4_3_con
v[0][0]']

 conv4_block5_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block4_out
[0][0]']
 tchNormalization)

 conv4_block5_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block5_preac
t_bn[0][0]
 Activation)                                                        ']

 conv4_block5_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block5_preac
t_relu[0][
 D)                                                                 0]']

 conv4_block5_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block5_1_con
v[0][0]']
 rmalization)

 conv4_block5_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block5_1_bn
[0][0]']
 ation)

 conv4_block5_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block5_1_rel
u[0][0]']
 dding2D)

 conv4_block5_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block5_2_pad
[0][0]']
 D)

 conv4_block5_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block5_2_con
v[0][0]']
 rmalization)

 conv4_block5_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block5_2_bn
[0][0]']
 ation)

 conv4_block5_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block5_2_rel
u[0][0]']
 D)
```

```
 conv4_block5_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block4_out
[0][0]',
                                                                     'conv4_block5_3_con
v[0][0]']

 conv4_block6_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block5_out
[0][0]']
 tchNormalization)

 conv4_block6_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block6_preac
t_bn[0][0]
 Activation)                                                         ']

 conv4_block6_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block6_preac
t_relu[0][
 D)                                                                  0]']

 conv4_block6_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block6_1_con
v[0][0]']
 rmalization)

 conv4_block6_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block6_1_bn
[0][0]']
 ation)

 conv4_block6_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block6_1_rel
u[0][0]']
 dding2D)

 conv4_block6_2_conv (Conv2  (None, 7, 7, 256)            589824    ['conv4_block6_2_pad
[0][0]']
 D)

 conv4_block6_2_bn (BatchNo  (None, 7, 7, 256)            1024      ['conv4_block6_2_con
v[0][0]']
 rmalization)

 conv4_block6_2_relu (Activ  (None, 7, 7, 256)            0         ['conv4_block6_2_bn
[0][0]']
 ation)

 max_pooling2d_2 (MaxPoolin  (None, 7, 7, 1024)           0         ['conv4_block5_out
[0][0]']
 g2D)
```

```
 conv4_block6_3_conv (Conv2  (None, 7, 7, 1024)          263168    ['conv4_block6_2_rel
u[0][0]']
 D)


 conv4_block6_out (Add)      (None, 7, 7, 1024)          0         ['max_pooling2d_2[0]
[0]',
                                                                   'conv4_block6_3_con
v[0][0]']

 conv5_block1_preact_bn (Ba  (None, 7, 7, 1024)          4096      ['conv4_block6_out
[0][0]']
 tchNormalization)


 conv5_block1_preact_relu (  (None, 7, 7, 1024)          0         ['conv5_block1_preac
t_bn[0][0]
 Activation)                                                       ']


 conv5_block1_1_conv (Conv2  (None, 7, 7, 512)           524288    ['conv5_block1_preac
t_relu[0][
 D)                                                               0]']


 conv5_block1_1_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block1_1_con
v[0][0]']
 rmalization)


 conv5_block1_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block1_1_bn
[0][0]']
 ation)


 conv5_block1_2_pad (ZeroPa  (None, 9, 9, 512)           0         ['conv5_block1_1_rel
u[0][0]']
 dding2D)


 conv5_block1_2_conv (Conv2  (None, 7, 7, 512)           2359296   ['conv5_block1_2_pad
[0][0]']
 D)


 conv5_block1_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block1_2_con
v[0][0]']
 rmalization)


 conv5_block1_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block1_2_bn
[0][0]']
 ation)
```

```
 conv5_block1_0_conv (Conv2  (None, 7, 7, 2048)          2099200   ['conv5_block1_preac
t_relu[0][
 D)                                                               0]']


 conv5_block1_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block1_2_rel
u[0][0]']
 D)


 conv5_block1_out (Add)      (None, 7, 7, 2048)          0         ['conv5_block1_0_con
v[0][0]',
                                                                   'conv5_block1_3_con
v[0][0]']


 conv5_block2_preact_bn (Ba  (None, 7, 7, 2048)          8192      ['conv5_block1_out
[0][0]']
 tchNormalization)


 conv5_block2_preact_relu (  (None, 7, 7, 2048)          0         ['conv5_block2_preac
t_bn[0][0]
 Activation)                                                       ']


 conv5_block2_1_conv (Conv2  (None, 7, 7, 512)           1048576   ['conv5_block2_preac
t_relu[0][
 D)                                                               0]']


 conv5_block2_1_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block2_1_con
v[0][0]']
 rmalization)


 conv5_block2_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block2_1_bn
[0][0]']
 ation)


 conv5_block2_2_pad (ZeroPa  (None, 9, 9, 512)           0         ['conv5_block2_1_rel
u[0][0]']
 dding2D)


 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)           2359296   ['conv5_block2_2_pad
[0][0]']
 D)


 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block2_2_con
v[0][0]']
 rmalization)
```

```
 conv5_block2_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block2_2_bn
[0][0]']
 ation)


 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block2_2_rel
u[0][0]']
 D)


 conv5_block2_out (Add)      (None, 7, 7, 2048)          0         ['conv5_block1_out
[0][0]',
                                                                    'conv5_block2_3_con
v[0][0]']


 conv5_block3_preact_bn (Ba  (None, 7, 7, 2048)          8192      ['conv5_block2_out
[0][0]']
 tchNormalization)


 conv5_block3_preact_relu (  (None, 7, 7, 2048)          0         ['conv5_block3_preac
t_bn[0][0]
 Activation)                                                        ']


 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)           1048576   ['conv5_block3_preac
t_relu[0][
 D)                                                                0]']


 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3_1_con
v[0][0]']
 rmalization)


 conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_1_bn
[0][0]']
 ation)


 conv5_block3_2_pad (ZeroPa  (None, 9, 9, 512)           0         ['conv5_block3_1_rel
u[0][0]']
 dding2D)


 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359296   ['conv5_block3_2_pad
[0][0]']
 D)


 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3_2_con
v[0][0]']
 rmalization)
```

```
 conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_2_bn
[0][0]']
 ation)


 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block3_2_rel
u[0][0]']
 D)


 conv5_block3_out (Add)      (None, 7, 7, 2048)          0         ['conv5_block2_out
[0][0]',
                                                                  'conv5_block3_3_con
v[0][0]']


 post_bn (BatchNormalizatio  (None, 7, 7, 2048)          8192      ['conv5_block3_out
[0][0]']
 n)


 post_relu (Activation)      (None, 7, 7, 2048)          0         ['post_bn[0][0]']


==========
Total params: 23564800 (89.89 MB)
Trainable params: 23519360 (89.72 MB)
Non-trainable params: 45440 (177.50 KB)
_________________________________________________________________
_________
```

We want to freeze all the layers except the last 50 layers.

In [ ]:
```
# Freezing all layers except last 50 layers

resnet.trainable = True

for layer in resnet.layers[:-50]:
    layer.trainable = False

resnet.summary()
```

```
Model: "resnet50v2"
_________________________________________________________________
_________
 Layer (type)                Output Shape              Param #   Connected to
==========
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         []


 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)       0         ['input_1[0][0]']


 conv1_conv (Conv2D)         (None, 112, 112, 64)      9472      ['conv1_pad[0][0]']
```

```
 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)         0         ['conv1_conv[0][0]']


 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)          0         ['pool1_pad[0][0]']


 conv2_block1_preact_bn (Ba  (None, 56, 56, 64)          256       ['pool1_pool[0][0]']

 tchNormalization)


 conv2_block1_preact_relu (  (None, 56, 56, 64)          0         ['conv2_block1_preac
t_bn[0][0]
 Activation)                                                       ']


 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)          4096      ['conv2_block1_preac
t_relu[0][
 D)                                                                0]']


 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)          256       ['conv2_block1_1_con
v[0][0]']
 rmalization)


 conv2_block1_1_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block1_1_bn
[0][0]']
 ation)


 conv2_block1_2_pad (ZeroPa  (None, 58, 58, 64)          0         ['conv2_block1_1_rel
u[0][0]']
 dding2D)


 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)          36864     ['conv2_block1_2_pad
[0][0]']
 D)


 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)          256       ['conv2_block1_2_con
v[0][0]']
 rmalization)


 conv2_block1_2_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block1_2_bn
[0][0]']
 ation)


 conv2_block1_0_conv (Conv2  (None, 56, 56, 256)         16640     ['conv2_block1_preac
t_relu[0][
 D)                                                                0]']
```

```
 conv2_block1_3_conv (Conv2  (None, 56, 56, 256)          16640     ['conv2_block1_2_rel
u[0][0]']
 D)

 conv2_block1_out (Add)      (None, 56, 56, 256)          0         ['conv2_block1_0_con
v[0][0]',
                                                                     'conv2_block1_3_con
v[0][0]']

 conv2_block2_preact_bn (Ba  (None, 56, 56, 256)          1024      ['conv2_block1_out
[0][0]']
 tchNormalization)

 conv2_block2_preact_relu (  (None, 56, 56, 256)          0         ['conv2_block2_preac
t_bn[0][0]
 Activation)                                                         ']

 conv2_block2_1_conv (Conv2  (None, 56, 56, 64)           16384     ['conv2_block2_preac
t_relu[0][
 D)                                                                  0']']

 conv2_block2_1_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block2_1_con
v[0][0]']
 rmalization)

 conv2_block2_1_relu (Activ  (None, 56, 56, 64)           0         ['conv2_block2_1_bn
[0][0]']
 ation)

 conv2_block2_2_pad (ZeroPa  (None, 58, 58, 64)           0         ['conv2_block2_1_rel
u[0][0]']
 dding2D)

 conv2_block2_2_conv (Conv2  (None, 56, 56, 64)           36864     ['conv2_block2_2_pad
[0][0]']
 D)

 conv2_block2_2_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block2_2_con
v[0][0]']
 rmalization)

 conv2_block2_2_relu (Activ  (None, 56, 56, 64)           0         ['conv2_block2_2_bn
[0][0]']
 ation)
```

```
 conv2_block2_3_conv (Conv2  (None, 56, 56, 256)          16640     ['conv2_block2_2_rel
u[0][0]']
 D)

 conv2_block2_out (Add)      (None, 56, 56, 256)          0         ['conv2_block1_out
[0][0]',
                                                                    'conv2_block2_3_con
v[0][0]']

 conv2_block3_preact_bn (Ba  (None, 56, 56, 256)          1024      ['conv2_block2_out
[0][0]']
 tchNormalization)

 conv2_block3_preact_relu (  (None, 56, 56, 256)          0         ['conv2_block3_preac
t_bn[0][0]
 Activation)                                                        ']

 conv2_block3_1_conv (Conv2  (None, 56, 56, 64)           16384     ['conv2_block3_preac
t_relu[0][
 D)                                                                 0]']

 conv2_block3_1_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block3_1_con
v[0][0]']
 rmalization)

 conv2_block3_1_relu (Activ  (None, 56, 56, 64)           0         ['conv2_block3_1_bn
[0][0]']
 ation)

 conv2_block3_2_pad (ZeroPa  (None, 58, 58, 64)           0         ['conv2_block3_1_rel
u[0][0]']
 dding2D)

 conv2_block3_2_conv (Conv2  (None, 28, 28, 64)           36864     ['conv2_block3_2_pad
[0][0]']
 D)

 conv2_block3_2_bn (BatchNo  (None, 28, 28, 64)           256       ['conv2_block3_2_con
v[0][0]']
 rmalization)

 conv2_block3_2_relu (Activ  (None, 28, 28, 64)           0         ['conv2_block3_2_bn
[0][0]']
 ation)
```

```
 max_pooling2d (MaxPooling2  (None, 28, 28, 256)          0         ['conv2_block2_out
[0][0]']
 D)


 conv2_block3_3_conv (Conv2  (None, 28, 28, 256)          16640     ['conv2_block3_2_rel
u[0][0]']
 D)


 conv2_block3_out (Add)      (None, 28, 28, 256)          0         ['max_pooling2d[0]
[0]',
                                                                     'conv2_block3_3_con
v[0][0]']

 conv3_block1_preact_bn (Ba  (None, 28, 28, 256)          1024      ['conv2_block3_out
[0][0]']
 tchNormalization)


 conv3_block1_preact_relu (  (None, 28, 28, 256)          0         ['conv3_block1_preac
t_bn[0][0]
 Activation)                                                         ']


 conv3_block1_1_conv (Conv2  (None, 28, 28, 128)          32768     ['conv3_block1_preac
t_relu[0][
 D)                                                                  0]']


 conv3_block1_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block1_1_con
v[0][0]']
 rmalization)


 conv3_block1_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block1_1_bn
[0][0]']
 ation)


 conv3_block1_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block1_1_rel
u[0][0]']
 dding2D)


 conv3_block1_2_conv (Conv2  (None, 28, 28, 128)          147456    ['conv3_block1_2_pad
[0][0]']
 D)


 conv3_block1_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block1_2_con
v[0][0]']
 rmalization)
```

```
 conv3_block1_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block1_2_bn
[0][0]']
 ation)


 conv3_block1_0_conv (Conv2  (None, 28, 28, 512)          131584    ['conv3_block1_preac
t_relu[0][
 D)                                                                 0]']


 conv3_block1_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block1_2_rel
u[0][0]']
 D)


 conv3_block1_out (Add)      (None, 28, 28, 512)          0         ['conv3_block1_0_con
v[0][0]',
                                                                    'conv3_block1_3_con
v[0][0]']


 conv3_block2_preact_bn (Ba  (None, 28, 28, 512)          2048      ['conv3_block1_out
[0][0]']
 tchNormalization)


 conv3_block2_preact_relu (  (None, 28, 28, 512)          0         ['conv3_block2_preac
t_bn[0][0]
 Activation)                                                        ']


 conv3_block2_1_conv (Conv2  (None, 28, 28, 128)          65536     ['conv3_block2_preac
t_relu[0][
 D)                                                                 0]']


 conv3_block2_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block2_1_con
v[0][0]']
 rmalization)


 conv3_block2_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block2_1_bn
[0][0]']
 ation)


 conv3_block2_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block2_1_rel
u[0][0]']
 dding2D)


 conv3_block2_2_conv (Conv2  (None, 28, 28, 128)          147456    ['conv3_block2_2_pad
[0][0]']
 D)
```

```
 conv3_block2_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block2_2_con
v[0][0]']
 rmalization)


 conv3_block2_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block2_2_bn
[0][0]']
 ation)


 conv3_block2_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block2_2_rel
u[0][0]']
 D)


 conv3_block2_out (Add)      (None, 28, 28, 512)          0         ['conv3_block1_out
[0][0]',
                                                                     'conv3_block2_3_con
v[0][0]']

 conv3_block3_preact_bn (Ba  (None, 28, 28, 512)          2048      ['conv3_block2_out
[0][0]']
 tchNormalization)


 conv3_block3_preact_relu (  (None, 28, 28, 512)          0         ['conv3_block3_preac
t_bn[0][0]
 Activation)                                                         ']


 conv3_block3_1_conv (Conv2  (None, 28, 28, 128)          65536     ['conv3_block3_preac
t_relu[0][
 D)                                                                  0]']


 conv3_block3_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block3_1_con
v[0][0]']
 rmalization)


 conv3_block3_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block3_1_bn
[0][0]']
 ation)


 conv3_block3_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block3_1_rel
u[0][0]']
 dding2D)


 conv3_block3_2_conv (Conv2  (None, 28, 28, 128)          147456    ['conv3_block3_2_pad
[0][0]']
 D)
```

```
 conv3_block3_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block3_2_con
v[0][0]']
 rmalization)


 conv3_block3_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block3_2_bn
[0][0]']
 ation)


 conv3_block3_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block3_2_rel
u[0][0]']
 D)


 conv3_block3_out (Add)      (None, 28, 28, 512)          0         ['conv3_block2_out
[0][0]',
                                                                     'conv3_block3_3_con
v[0][0]']

 conv3_block4_preact_bn (Ba  (None, 28, 28, 512)          2048      ['conv3_block3_out
[0][0]']
 tchNormalization)


 conv3_block4_preact_relu (  (None, 28, 28, 512)          0         ['conv3_block4_preac
t_bn[0][0]
 Activation)                                                         ']


 conv3_block4_1_conv (Conv2  (None, 28, 28, 128)          65536     ['conv3_block4_preac
t_relu[0][
 D)                                                                  0]']


 conv3_block4_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block4_1_con
v[0][0]']
 rmalization)


 conv3_block4_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block4_1_bn
[0][0]']
 ation)


 conv3_block4_2_pad (ZeroPa  (None, 30, 30, 128)          0         ['conv3_block4_1_rel
u[0][0]']
 dding2D)


 conv3_block4_2_conv (Conv2  (None, 14, 14, 128)          147456    ['conv3_block4_2_pad
[0][0]']
 D)
```

```
 conv3_block4_2_bn (BatchNo  (None, 14, 14, 128)          512       ['conv3_block4_2_con
v[0][0]']
 rmalization)


 conv3_block4_2_relu (Activ  (None, 14, 14, 128)          0         ['conv3_block4_2_bn
[0][0]']
 ation)


 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 512)          0         ['conv3_block3_out
[0][0]']
 g2D)


 conv3_block4_3_conv (Conv2  (None, 14, 14, 512)          66048     ['conv3_block4_2_rel
u[0][0]']
 D)


 conv3_block4_out (Add)      (None, 14, 14, 512)          0         ['max_pooling2d_1[0]
[0]',
                                                                     'conv3_block4_3_con
v[0][0]']

 conv4_block1_preact_bn (Ba  (None, 14, 14, 512)          2048      ['conv3_block4_out
[0][0]']
 tchNormalization)


 conv4_block1_preact_relu (  (None, 14, 14, 512)          0         ['conv4_block1_preac
t_bn[0][0]
 Activation)                                                         ']


 conv4_block1_1_conv (Conv2  (None, 14, 14, 256)          131072    ['conv4_block1_preac
t_relu[0][
 D)                                                                  0]']


 conv4_block1_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block1_1_con
v[0][0]']
 rmalization)


 conv4_block1_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block1_1_bn
[0][0]']
 ation)


 conv4_block1_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block1_1_rel
u[0][0]']
 dding2D)
```

```
 conv4_block1_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block1_2_pad
[0][0]']
 D)


 conv4_block1_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block1_2_con
v[0][0]']
 rmalization)


 conv4_block1_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block1_2_bn
[0][0]']
 ation)


 conv4_block1_0_conv (Conv2  (None, 14, 14, 1024)         525312    ['conv4_block1_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block1_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block1_2_rel
u[0][0]']
 D)


 conv4_block1_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block1_0_con
v[0][0]',
                                                                    'conv4_block1_3_con
v[0][0]']

 conv4_block2_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block1_out
[0][0]']
 tchNormalization)


 conv4_block2_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block2_preac
t_bn[0][0]
 Activation)                                                        ']


 conv4_block2_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block2_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block2_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block2_1_con
v[0][0]']
 rmalization)


 conv4_block2_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block2_1_bn
[0][0]']
 ation)
```

```
 conv4_block2_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block2_1_rel
u[0][0]']
 dding2D)


 conv4_block2_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block2_2_pad
[0][0]']
 D)


 conv4_block2_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block2_2_con
v[0][0]']
 rmalization)


 conv4_block2_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block2_2_bn
[0][0]']
 ation)


 conv4_block2_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block2_2_rel
u[0][0]']
 D)


 conv4_block2_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block1_out
[0][0]',
                                                                     'conv4_block2_3_con
v[0][0]']

 conv4_block3_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block2_out
[0][0]']
 tchNormalization)


 conv4_block3_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block3_preac
t_bn[0][0]
 Activation)                                                        ']


 conv4_block3_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block3_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block3_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block3_1_con
v[0][0]']
 rmalization)


 conv4_block3_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block3_1_bn
[0][0]']
 ation)
```

```
 conv4_block3_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block3_1_rel
u[0][0]']
 dding2D)


 conv4_block3_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block3_2_pad
[0][0]']
 D)


 conv4_block3_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block3_2_con
v[0][0]']
 rmalization)


 conv4_block3_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block3_2_bn
[0][0]']
 ation)


 conv4_block3_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block3_2_rel
u[0][0]']
 D)


 conv4_block3_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block2_out
[0][0]',
                                                                     'conv4_block3_3_con
v[0][0]']

 conv4_block4_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block3_out
[0][0]']
 tchNormalization)


 conv4_block4_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block4_preac
t_bn[0][0]
 Activation)                                                        ']


 conv4_block4_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block4_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block4_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block4_1_con
v[0][0]']
 rmalization)


 conv4_block4_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block4_1_bn
[0][0]']
 ation)
```

```
 conv4_block4_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block4_1_rel
u[0][0]']
 dding2D)


 conv4_block4_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block4_2_pad
[0][0]']
 D)


 conv4_block4_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block4_2_con
v[0][0]']
 rmalization)


 conv4_block4_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block4_2_bn
[0][0]']
 ation)


 conv4_block4_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block4_2_rel
u[0][0]']
 D)


 conv4_block4_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block3_out
[0][0]',
                                                                     'conv4_block4_3_con
v[0][0]']

 conv4_block5_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block4_out
[0][0]']
 tchNormalization)


 conv4_block5_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block5_preac
t_bn[0][0]
 Activation)                                                        ']


 conv4_block5_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block5_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block5_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block5_1_con
v[0][0]']
 rmalization)


 conv4_block5_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block5_1_bn
[0][0]']
 ation)
```

```
 conv4_block5_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block5_1_rel
u[0][0]']
 dding2D)


 conv4_block5_2_conv (Conv2  (None, 14, 14, 256)          589824    ['conv4_block5_2_pad
[0][0]']
 D)


 conv4_block5_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block5_2_con
v[0][0]']
 rmalization)


 conv4_block5_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block5_2_bn
[0][0]']
 ation)


 conv4_block5_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block5_2_rel
u[0][0]']
 D)


 conv4_block5_out (Add)      (None, 14, 14, 1024)         0         ['conv4_block4_out
[0][0]',
                                                                     'conv4_block5_3_con
v[0][0]']

 conv4_block6_preact_bn (Ba  (None, 14, 14, 1024)         4096      ['conv4_block5_out
[0][0]']
 tchNormalization)


 conv4_block6_preact_relu (  (None, 14, 14, 1024)         0         ['conv4_block6_preac
t_bn[0][0]
 Activation)                                                        ']


 conv4_block6_1_conv (Conv2  (None, 14, 14, 256)          262144    ['conv4_block6_preac
t_relu[0][
 D)                                                                 0]']


 conv4_block6_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block6_1_con
v[0][0]']
 rmalization)


 conv4_block6_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block6_1_bn
[0][0]']
 ation)
```

```
 conv4_block6_2_pad (ZeroPa  (None, 16, 16, 256)          0         ['conv4_block6_1_rel
u[0][0]']
 dding2D)


 conv4_block6_2_conv (Conv2  (None, 7, 7, 256)            589824    ['conv4_block6_2_pad
[0][0]']
 D)


 conv4_block6_2_bn (BatchNo  (None, 7, 7, 256)            1024      ['conv4_block6_2_con
v[0][0]']
 rmalization)


 conv4_block6_2_relu (Activ  (None, 7, 7, 256)            0         ['conv4_block6_2_bn
[0][0]']
 ation)


 max_pooling2d_2 (MaxPoolin  (None, 7, 7, 1024)           0         ['conv4_block5_out
[0][0]']
 g2D)


 conv4_block6_3_conv (Conv2  (None, 7, 7, 1024)           263168    ['conv4_block6_2_rel
u[0][0]']
 D)


 conv4_block6_out (Add)      (None, 7, 7, 1024)           0         ['max_pooling2d_2[0]
[0]',
                                                                     'conv4_block6_3_con
v[0][0]']

 conv5_block1_preact_bn (Ba  (None, 7, 7, 1024)           4096      ['conv4_block6_out
[0][0]']
 tchNormalization)


 conv5_block1_preact_relu (  (None, 7, 7, 1024)           0         ['conv5_block1_preac
t_bn[0][0]
 Activation)                                                         ']


 conv5_block1_1_conv (Conv2  (None, 7, 7, 512)            524288    ['conv5_block1_preac
t_relu[0][
 D)                                                                  0]']


 conv5_block1_1_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block1_1_con
v[0][0]']
 rmalization)
```

```
 conv5_block1_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block1_1_bn
[0][0]']
 ation)


 conv5_block1_2_pad (ZeroPa  (None, 9, 9, 512)           0         ['conv5_block1_1_rel
u[0][0]']
 dding2D)


 conv5_block1_2_conv (Conv2  (None, 7, 7, 512)           2359296   ['conv5_block1_2_pad
[0][0]']
 D)


 conv5_block1_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block1_2_con
v[0][0]']
 rmalization)


 conv5_block1_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block1_2_bn
[0][0]']
 ation)


 conv5_block1_0_conv (Conv2  (None, 7, 7, 2048)          2099200   ['conv5_block1_preac
t_relu[0][
 D)                                                               0]']


 conv5_block1_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block1_2_rel
u[0][0]']
 D)


 conv5_block1_out (Add)      (None, 7, 7, 2048)          0         ['conv5_block1_0_con
v[0][0]',
                                                                  'conv5_block1_3_con
v[0][0]']


 conv5_block2_preact_bn (Ba  (None, 7, 7, 2048)          8192      ['conv5_block1_out
[0][0]']
 tchNormalization)


 conv5_block2_preact_relu (  (None, 7, 7, 2048)          0         ['conv5_block2_preac
t_bn[0][0]
 Activation)                                                      ']


 conv5_block2_1_conv (Conv2  (None, 7, 7, 512)           1048576   ['conv5_block2_preac
t_relu[0][
 D)                                                               0]']
```

```
 conv5_block2_1_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block2_1_con
v[0][0]']
 rmalization)

 conv5_block2_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block2_1_bn
[0][0]']
 ation)

 conv5_block2_2_pad (ZeroPa  (None, 9, 9, 512)           0         ['conv5_block2_1_rel
u[0][0]']
 dding2D)

 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)           2359296   ['conv5_block2_2_pad
[0][0]']
 D)

 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block2_2_con
v[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block2_2_bn
[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block2_2_rel
u[0][0]']
 D)

 conv5_block2_out (Add)      (None, 7, 7, 2048)          0         ['conv5_block1_out
[0][0]',
                                                                  'conv5_block2_3_con
v[0][0]']

 conv5_block3_preact_bn (Ba  (None, 7, 7, 2048)          8192      ['conv5_block2_out
[0][0]']
 tchNormalization)

 conv5_block3_preact_relu (  (None, 7, 7, 2048)          0         ['conv5_block3_preac
t_bn[0][0]
 Activation)                                                      ']

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)           1048576   ['conv5_block3_preac
t_relu[0][
 D)                                                               0]']
```

```
 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3_1_con
v[0][0]']
 rmalization)


 conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_1_bn
[0][0]']
 ation)


 conv5_block3_2_pad (ZeroPa  (None, 9, 9, 512)           0         ['conv5_block3_1_rel
u[0][0]']
 dding2D)


 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359296   ['conv5_block3_2_pad
[0][0]']
 D)


 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3_2_con
v[0][0]']
 rmalization)


 conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_2_bn
[0][0]']
 ation)


 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block3_2_rel
u[0][0]']
 D)


 conv5_block3_out (Add)      (None, 7, 7, 2048)          0         ['conv5_block2_out
[0][0]',
                                                                  'conv5_block3_3_con
v[0][0]']


 post_bn (BatchNormalizatio  (None, 7, 7, 2048)          8192      ['conv5_block3_out
[0][0]']
 n)


 post_relu (Activation)      (None, 7, 7, 2048)          0         ['post_bn[0][0]']


==================================================================================
==========
Total params: 23564800 (89.89 MB)
Trainable params: 16352256 (62.38 MB)
Non-trainable params: 7212544 (27.51 MB)
```

In [ ]:
```
def Create_ResNet50V2_Model():

    model = Sequential([
                        resnet,
                        Dropout(0.25),
                        BatchNormalization(),
                        Flatten(),
                        Dense(64, activation='relu'),
                        BatchNormalization(),
                        Dropout(0.5),
                        Dense(7,activation='softmax')
                       ])
    return model
```

In [ ]:
```
# Creating the final model
model = Create_ResNet50V2_Model()

# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 dropout (Dropout)           (None, 7, 7, 2048)        0

 batch_normalization (Batch  (None, 7, 7, 2048)        8192
 Normalization)

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 64)                6422592

 batch_normalization_1 (Bat  (None, 64)                256
 chNormalization)

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 7)                 455

=================================================================
Total params: 29996295 (114.43 MB)
Trainable params: 22779527 (86.90 MB)
Non-trainable params: 7216768 (27.53 MB)
_________________________________________________________________
```

4. Callbacks

In [ ]:
```
# File path for the model checkpoint
cnn_path = '/content/FER_2013_Emotion_Classifier/ResNet50_Transfer_Learning'
name = 'ResNet50_Transfer_Learning.keras'
chk_path = os.path.join(cnn_path, name)

# Callback to save the model checkpoint
checkpoint = ModelCheckpoint(filepath=chk_path,
                             save_best_only=True,
                             verbose=1,
                             mode='min',
                             monitor='val_loss')
```

```
# Callback for early stopping
earlystop = EarlyStopping(monitor='val_accuracy',
                          patience=7,
                          verbose=1,
                          restore_best_weights=True)

# Callback to reduce learning rate
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=2,
                              verbose=1)

# Callback to log training data to a CSV file
csv_logger = CSVLogger(os.path.join(cnn_path,'training.log'))

# Aggregating all callbacks into a list
# callbacks = [checkpoint, earlystop, reduce_lr, csv_logger]  # Adjusted as per your use
callbacks = [checkpoint, earlystop, csv_logger]  # Adjusted as per your use-case
```

5. Training Model

In [ ]:
```
# Mount Google Drive
from google.colab import drive
drive.mount('/gdrive')
```

```
Mounted at /gdrive
```

In [ ]:
```
# Upload best ResNet50 model
import keras

model = keras.saving.load_model("/gdrive/MyDrive/ancilcleetus-github/My-Learning-Journey
```

In [ ]:
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [ ]:
```
# Calculate steps per epoch (no of batches needed to finish 1 epoch)
train_steps_per_epoch = np.ceil(train_generator.samples / train_generator.batch_size)
validation_steps_per_epoch = np.ceil(validation_generator.samples / validation_generator
test_steps_per_epoch = np.ceil(test_generator.samples / test_generator.batch_size)
print(f"train_steps_per_epoch = {train_steps_per_epoch}")
print(f"validation_steps_per_epoch = {validation_steps_per_epoch}")
print(f"test_steps_per_epoch = {test_steps_per_epoch}")
```

```
train_steps_per_epoch = 359.0
validation_steps_per_epoch = 90.0
test_steps_per_epoch = 113.0
```

In [ ]:
```
history = model.fit(
    train_generator,
    steps_per_epoch=train_steps_per_epoch,
    epochs=40,
    validation_data=validation_generator,
    validation_steps=validation_steps_per_epoch,
    class_weight=class_weights_dict,
    callbacks=callbacks)
```

```
Epoch 1/40
359/359 [==============================] - ETA: 0s - loss: 0.8909 - accuracy: 0.6538
Epoch 1: val_loss improved from inf to 1.07893, saving model to /content/FER_2013_Emotio
n_Classifier/ResNet50_Transfer_Learning/ResNet50_Transfer_Learning.keras
359/359 [==============================] - 490s 1s/step - loss: 0.8909 - accuracy: 0.653
8 - val_loss: 1.0789 - val_accuracy: 0.6114
Epoch 2/40
```

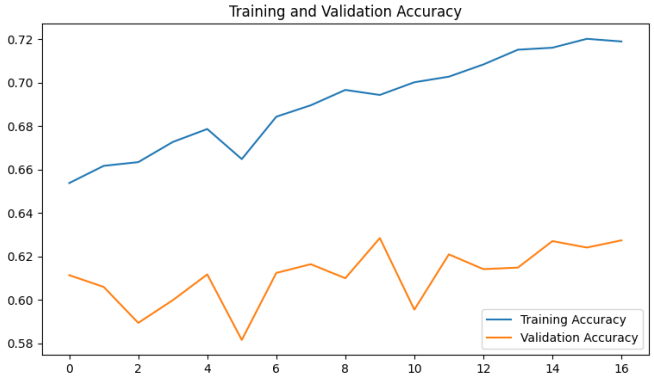
```
359/359 [==============================] - ETA: 0s - loss: 0.8719 - accuracy: 0.6617
Epoch 2: val_loss improved from 1.07893 to 1.07485, saving model to /content/FER_2013_Em
otion_Classifier/ResNet50_Transfer_Learning/ResNet50_Transfer_Learning.keras
359/359 [==============================] - 406s 1s/step - loss: 0.8719 - accuracy: 0.661
7 - val_loss: 1.0748 - val_accuracy: 0.6060
Epoch 3/40
359/359 [==============================] - ETA: 0s - loss: 0.8616 - accuracy: 0.6634
Epoch 3: val_loss did not improve from 1.07485
359/359 [==============================] - 401s 1s/step - loss: 0.8616 - accuracy: 0.663
4 - val_loss: 1.1605 - val_accuracy: 0.5894
Epoch 4/40
359/359 [==============================] - ETA: 0s - loss: 0.8331 - accuracy: 0.6727
Epoch 4: val_loss did not improve from 1.07485
359/359 [==============================] - 399s 1s/step - loss: 0.8331 - accuracy: 0.672
7 - val_loss: 1.0936 - val_accuracy: 0.5999
Epoch 5/40
359/359 [==============================] - ETA: 0s - loss: 0.8175 - accuracy: 0.6786
Epoch 5: val_loss improved from 1.07485 to 1.06427, saving model to /content/FER_2013_Em
otion_Classifier/ResNet50_Transfer_Learning/ResNet50_Transfer_Learning.keras
359/359 [==============================] - 404s 1s/step - loss: 0.8175 - accuracy: 0.678
6 - val_loss: 1.0643 - val_accuracy: 0.6117
Epoch 6/40
359/359 [==============================] - ETA: 0s - loss: 0.8582 - accuracy: 0.6648
Epoch 6: val_loss did not improve from 1.06427
359/359 [==============================] - 404s 1s/step - loss: 0.8582 - accuracy: 0.664
8 - val_loss: 1.1537 - val_accuracy: 0.5816
Epoch 7/40
359/359 [==============================] - ETA: 0s - loss: 0.8048 - accuracy: 0.6843
Epoch 7: val_loss improved from 1.06427 to 1.05054, saving model to /content/FER_2013_Em
otion_Classifier/ResNet50_Transfer_Learning/ResNet50_Transfer_Learning.keras
359/359 [==============================] - 411s 1s/step - loss: 0.8048 - accuracy: 0.684
3 - val_loss: 1.0505 - val_accuracy: 0.6124
Epoch 8/40
359/359 [==============================] - ETA: 0s - loss: 0.7856 - accuracy: 0.6896
Epoch 8: val_loss did not improve from 1.05054
359/359 [==============================] - 406s 1s/step - loss: 0.7856 - accuracy: 0.689
6 - val_loss: 1.0598 - val_accuracy: 0.6164
Epoch 9/40
359/359 [==============================] - ETA: 0s - loss: 0.7706 - accuracy: 0.6966
Epoch 9: val_loss did not improve from 1.05054
359/359 [==============================] - 407s 1s/step - loss: 0.7706 - accuracy: 0.696
6 - val_loss: 1.0737 - val_accuracy: 0.6100
Epoch 10/40
359/359 [==============================] - ETA: 0s - loss: 0.7726 - accuracy: 0.6943
Epoch 10: val_loss did not improve from 1.05054
359/359 [==============================] - 409s 1s/step - loss: 0.7726 - accuracy: 0.694
3 - val_loss: 1.0549 - val_accuracy: 0.6285
Epoch 11/40
359/359 [==============================] - ETA: 0s - loss: 0.7631 - accuracy: 0.7001
Epoch 11: val_loss did not improve from 1.05054
359/359 [==============================] - 407s 1s/step - loss: 0.7631 - accuracy: 0.700
1 - val_loss: 1.1267 - val_accuracy: 0.5955
Epoch 12/40
359/359 [==============================] - ETA: 0s - loss: 0.7496 - accuracy: 0.7027
Epoch 12: val_loss improved from 1.05054 to 1.05033, saving model to /content/FER_2013_E
motion_Classifier/ResNet50_Transfer_Learning/ResNet50_Transfer_Learning.keras
359/359 [==============================] - 410s 1s/step - loss: 0.7496 - accuracy: 0.702
7 - val_loss: 1.0503 - val_accuracy: 0.6210
Epoch 13/40
359/359 [==============================] - ETA: 0s - loss: 0.7463 - accuracy: 0.7083
Epoch 13: val_loss did not improve from 1.05033
359/359 [==============================] - 402s 1s/step - loss: 0.7463 - accuracy: 0.708
3 - val_loss: 1.1115 - val_accuracy: 0.6142
Epoch 14/40
359/359 [==============================] - ETA: 0s - loss: 0.7199 - accuracy: 0.7151
Epoch 14: val_loss did not improve from 1.05033
```

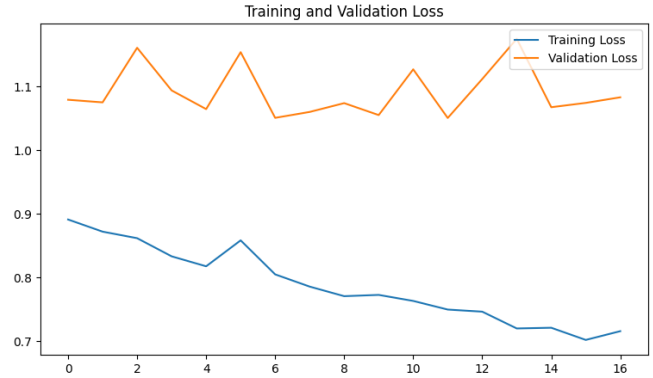
```
359/359 [==============================] - 397s 1s/step - loss: 0.7199 - accuracy: 0.715
1 - val_loss: 1.1746 - val_accuracy: 0.6149
Epoch 15/40
359/359 [==============================] - ETA: 0s - loss: 0.7210 - accuracy: 0.7160
Epoch 15: val_loss did not improve from 1.05033
359/359 [==============================] - 398s 1s/step - loss: 0.7210 - accuracy: 0.716
0 - val_loss: 1.0672 - val_accuracy: 0.6271
Epoch 16/40
359/359 [==============================] - ETA: 0s - loss: 0.7019 - accuracy: 0.7201
Epoch 16: val_loss did not improve from 1.05033
359/359 [==============================] - 395s 1s/step - loss: 0.7019 - accuracy: 0.720
1 - val_loss: 1.0740 - val_accuracy: 0.6241
Epoch 17/40
359/359 [==============================] - ETA: 0s - loss: 0.7157 - accuracy: 0.7189
Epoch 17: val_loss did not improve from 1.05033
Restoring model weights from the end of the best epoch: 10.
359/359 [==============================] - 405s 1s/step - loss: 0.7157 - accuracy: 0.718
9 - val_loss: 1.0828 - val_accuracy: 0.6274
Epoch 17: early stopping
```

6. Plotting Performance Metrics

In [ ]:
```python
def plot_training_history(history, save_path=None):
    """
    Plots the training and validation accuracy and loss.

    Parameters:
    - history: A Keras History object. Contains the logs from the training process.

    Returns:
    - None. Displays the matplotlib plots for training/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 = range(len(acc))

    plt.figure(figsize=(20, 5))

    # Plot training and validation accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    # Plot training and validation loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')

    if save_path:
        plt.savefig(save_path)

    plt.show()
```

In [ ]:
```python
training_history_save_path = '/content/FER_2013_Emotion_Classifier/ResNet50_Transfer_Lea
plot_training_history(history, training_history_save_path)
```





7. Model Evaluation

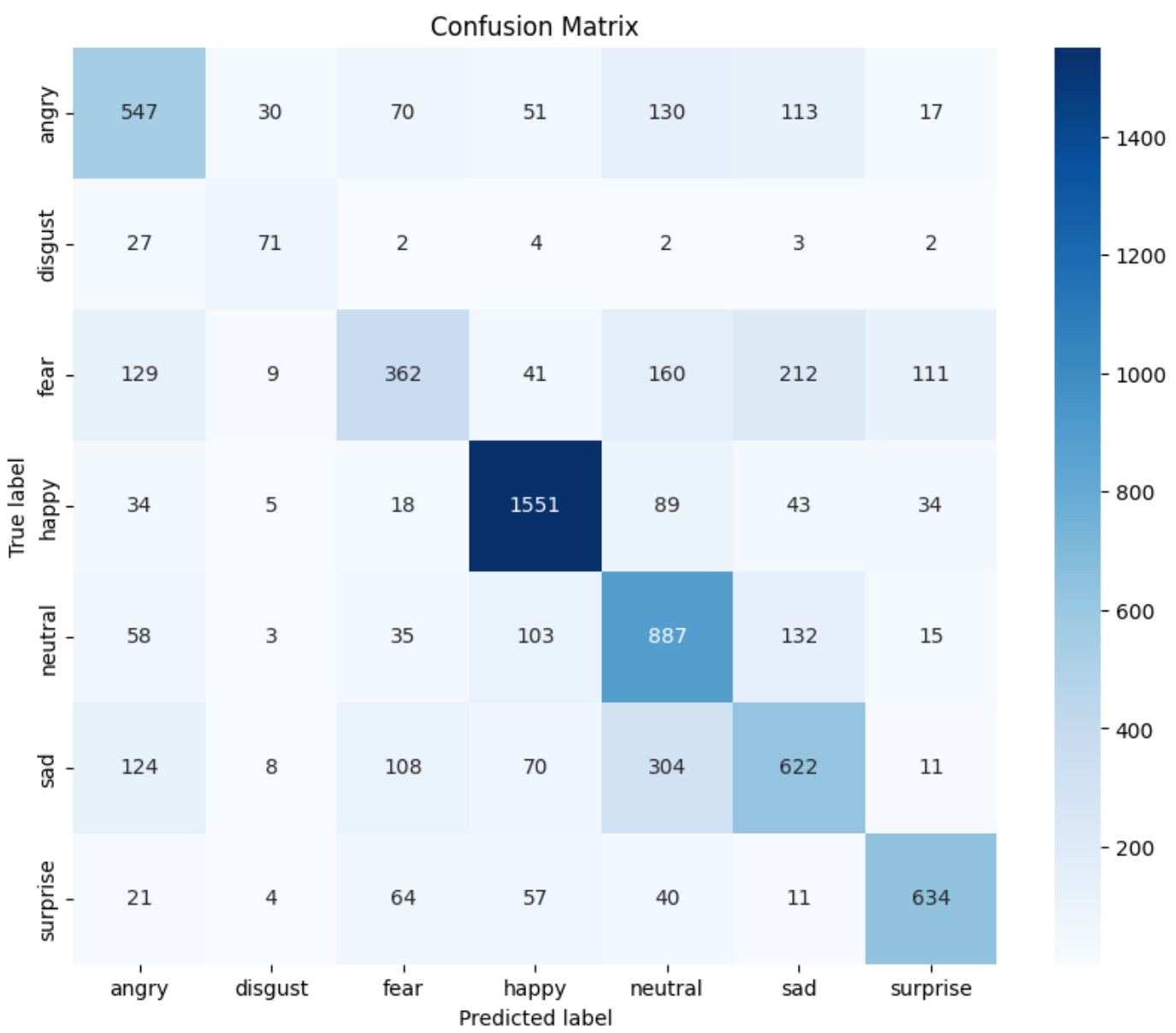
In [ ]:
```
train_loss, train_accu = model.evaluate(train_generator)
test_loss, test_accu = model.evaluate(test_generator)
print(f"Train accuracy = {train_accu*100:.2f} , Test accuracy = {test_accu*100:.2f}")
```

```
359/359 [==============================] - 309s 860ms/step - loss: 0.7630 - accuracy: 0.
7175
113/113 [==============================] - 22s 194ms/step - loss: 1.0093 - accuracy: 0.6
512
Train accuracy = 71.75 , Test accuracy = 65.12
```

8. Confusion Matrix

In [ ]:
```
# Assuming your true_classes and predicted_classes are already defined
true_classes = test_generator.classes
predicted_classes = np.argmax(model.predict(test_generator, steps=np.ceil(test_generator
class_labels = list(test_generator.class_indices.keys())

# Generate the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Plotting with seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')

confusion_matrix_save_path = '/content/FER_2013_Emotion_Classifier/ResNet50_Transfer_Lea
plt.savefig(confusion_matrix_save_path)

plt.show()
```

```
113/113 [==============================] - 26s 221ms/step
```



9. Classification Report

In [ ]:

```python
def save_classification_report(true_classes, predicted_classes, class_labels, save_path)
    """
    Generates a classification report and saves it as a PNG image.

    Parameters:
    - true_classes: Ground truth (correct) target values.
    - predicted_classes: Estimated targets as returned by a classifier.
    - class_labels: List of labels to index the matrix.
    - save_path: The path where the image will be saved.

    Returns:
    - None. Saves the classification report as a PNG image at the specified path.
    """
    report = classification_report(true_classes, predicted_classes, target_names=class_l

    # Create a matplotlib figure
    plt.figure(figsize=(10, 6))
    plt.text(0.01, 0.05, str(report), {'fontsize': 10}, fontproperties='monospace')
    plt.axis('off')

    # Save the figure
    plt.savefig(save_path, bbox_inches='tight', pad_inches=0.1)
```

```
    return report
```

In [ ]:
```
# Get Classification Report
classification_report_save_path = '/content/FER_2013_Emotion_Classifier/ResNet50_Transfe
report = save_classification_report(true_classes, predicted_classes, class_labels, class
print(f"Classification Report:\n{report}")
```

```
Classification Report:
              precision    recall  f1-score   support

       angry       0.58      0.57      0.58       958
     disgust       0.55      0.64      0.59       111
        fear       0.55      0.35      0.43      1024
       happy       0.83      0.87      0.85      1774
     neutral       0.55      0.72      0.62      1233
         sad       0.55      0.50      0.52      1247
    surprise       0.77      0.76      0.77       831

    accuracy                           0.65      7178
   macro avg       0.62      0.63      0.62      7178
weighted avg       0.65      0.65      0.64      7178
```


```
              precision    recall  f1-score   support

       angry       0.58      0.57      0.58       958
     disgust       0.55      0.64      0.59       111
        fear       0.55      0.35      0.43      1024
       happy       0.83      0.87      0.85      1774
     neutral       0.55      0.72      0.62      1233
         sad       0.55      0.50      0.52      1247
    surprise       0.77      0.76      0.77       831

    accuracy                           0.65      7178
   macro avg       0.62      0.63      0.62      7178
weighted avg       0.65      0.65      0.64      7178
```


Here's a breakdown of what each metric means in the above Classification Report:

- **Accuracy**
  - The overall proportion of correct predictions made by your model across all classes
  - Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$ where
    - TP $\implies$ No of correctly classified positive cases
    - FN $\implies$ No of positive cases incorrectly classified as negative
    - FP $\implies$ No of negative cases incorrectly classified as positive
    - TN $\implies$ No of correctly classified negative cases
- **Precision**
  - Measures the proportion of your predicted positives that were actually correct

    - Precision = $\frac{TP}{TP+FP}$
- **Recall**
    - Measures the proportion of actual positives that were correctly identified by your model
    - Recall = $\frac{TP}{TP+FN}$
- **F1-Score**
    - Harmonic mean between precision and recall, combining both metrics into a single score
    - F1-Score = $\frac{2 \times Precision \times Recall}{Precision+Recall}$
    - A high F1-score indicates that your model is both precise (few incorrect positives i.e. few false positives) and has good recall (identifies most of the actual positives i.e. few false negatives).
- **Support**
    - Refers to the total number of true instances for a particular class in your data
    - Support for a particular class = Sum of True labels for that class in the Confusion Matrix
- **Total Support**
    - Sum of supports across all classes
- **Macro Avg**
    - This is the unweighted average of the metric (precision, recall, F1-score) across all classes.
    - It can be useful when you have imbalanced data, where some classes have many more instances than others.
    - Macro Avg = $\frac{MetricClass1+MetricClass2+...+MetricClassN}{N}$ where N is the no of classes
- **Weighted Avg**
    - This is the weighted average of the metric (precision, recall, F1-score) across all classes, where the weight for each class is its support (the number of instances in that class).
    - This is useful when you care more about the performance of your model on the more frequent classes.
    - Weighted Avg = $\frac{SupportClass1 \times MetricClass1+SupportClass2 \times MetricClass2+...+SupportClassN \times MetricClassN}{TotalSupport}$

10. ROC Plots for all classes

1. **ROC Curves**:

    - ROC (Receiver Operating Characteristic) curves are a visualization tool typically used for binary classification problems. They plot the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis. TPR measures how well the model catches true positive cases, while FPR shows how often it mistakenly classifies negative cases as positive. ROC curves show how well the model can distinguish between positive and negative cases at various thresholds.

    - TPR = $\frac{TP}{TP+FN}$ where

        - TP $\implies$ No of correctly classified positive cases
        - FN $\implies$ No of positive cases incorrectly classified as negative
    - FPR = $\frac{FP}{FP+TN}$ where

        - FP $\implies$ No of negative cases incorrectly classified as positive
        - TN $\implies$ No of correctly classified negative cases
2. **Interpreting the Curve**:

- Ideal Curve:

  A perfect model would have a curve hugging the top-left corner of the graph. This indicates a high TPR (catching most positive cases) with a low FPR (minimal false positives).

- Diagonal Line:

  A random classifier would have a curve following the diagonal line (FPR = TPR). This means the model is no better than flipping a coin.

- Distance from Diagonal:

  The farther the curve is from the diagonal, the better the model can distinguish between classes.

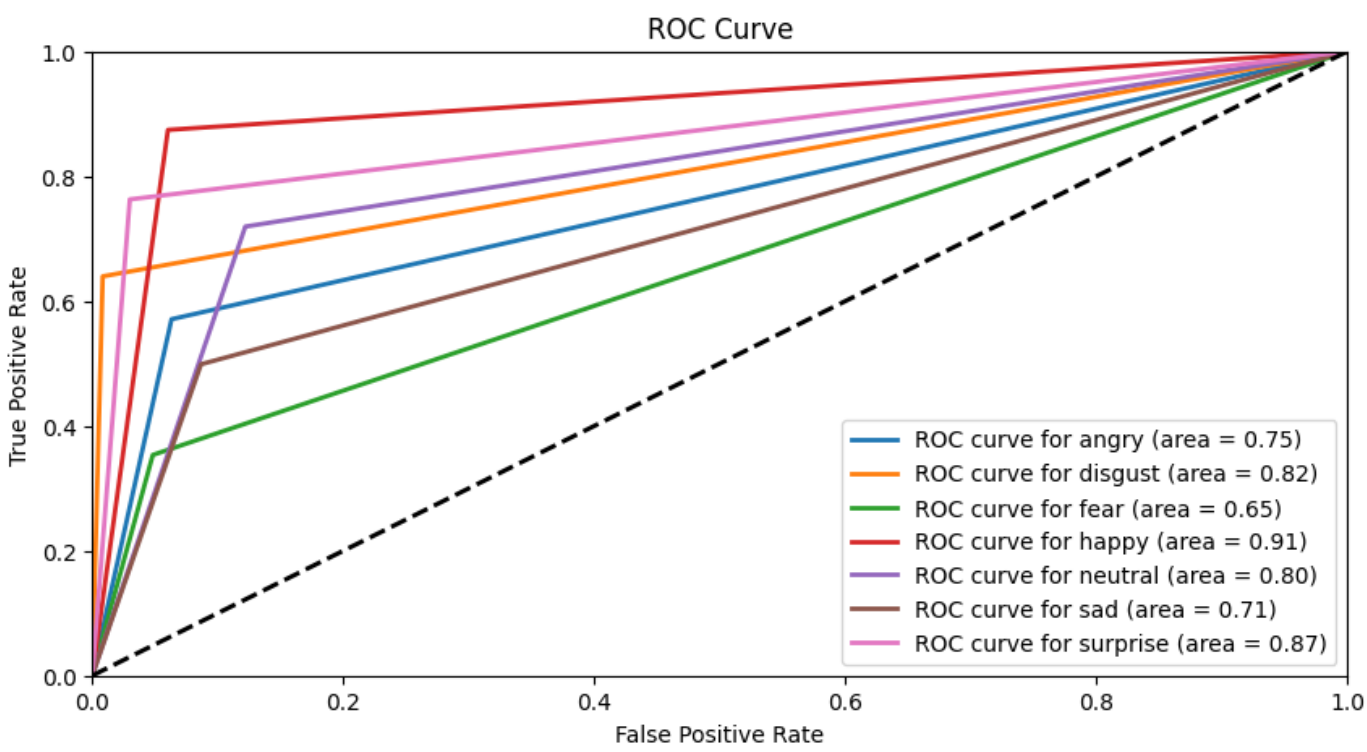
3. **AUC (Area Under Curve)**:

- More AUC for each class $\implies$ model has learnt really well & is able to predict accurately

In [ ]:
```
true_labels = test_generator.classes
preds = model.predict(test_generator, steps=len(test_generator))
pred_labels = np.argmax(preds, axis=1)
classes=list(test_generator.class_indices.keys())
```

```
113/113 [==============================] - 22s 192ms/step
```

In [ ]:
```
y_encoded = pd.get_dummies(true_labels).astype(int).values
preds_encoded = pd.get_dummies(pred_labels).astype(int).values

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(7):
    fpr[i], tpr[i], _ = roc_curve(y_encoded[:,i], preds_encoded[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(10,5))
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2']
for i, color in enumerate(colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f"ROC curve for {classes[i]} (area

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')

roc_curve_save_path = '/content/FER_2013_Emotion_Classifier/ResNet50_Transfer_Learning/r
plt.savefig(roc_curve_save_path)
```



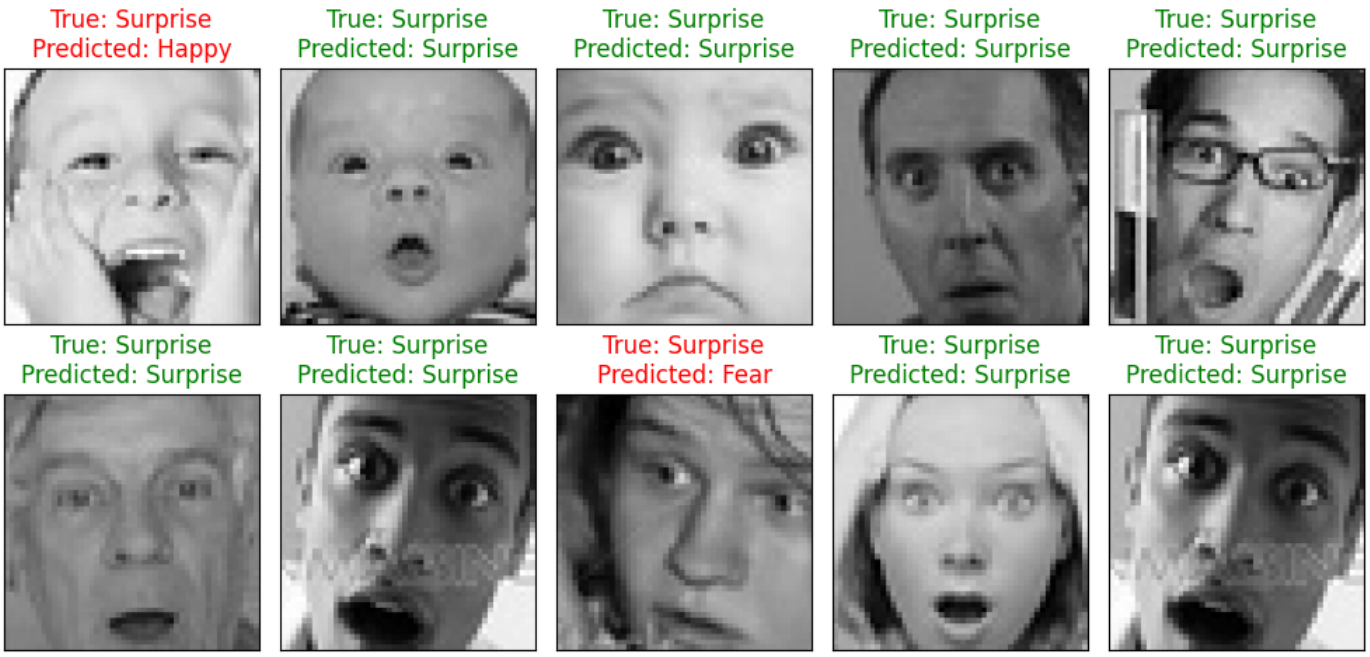
AUC for all classes are really good. Class `happy` has AUC of 0.91 while `surprise` has an AUC of 0.87. Even though `disgust` is the least represented class in the dataset, `disgust` has got an AUC of 0.82 which is really good.

We can use Confusion Matrix, Classification Report (with Precision, Recall, F1-Score), ROC Curve and AUC to quantify the performance of our model on all of these classes.

11. Making Predictions

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

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

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(test_generator) - 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_generator[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(test_generator[Random_batch][1][Random_Img_Index[i]], a

    # Making a prediction using the model
    Model_Prediction = np.argmax(model.predict(tf.expand_dims(Random_Img, axis=0), verbo

    # 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_Predic
```

```
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\nPredicted: {Emotion_Classe

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



12. Conclusion

We can make the following observations:

- We got 65.12 % Overall Accuracy on test set.
- We have very high values across diagonal cells of the test data confusion matrix in comparison to off-diagonal cells.
- We can see good values of Precision, Recall and F1-Score from the Classification Report.
- ROC curve is away from diagonal line corresponding to random classifier. Also, ROC curve is leaning towards the top-left corner of the graph.
- AUC for all classes are really good.
- Compared to all the different models tried earlier, we can see that our ResNet50 model has few misclassifications in the random test batch above.

Hence from all these observations, we can draw the conclusion that our ResNet50 model is good. At last, Transfer Learning with ResNet50 yielded good results. Now, we can focus on deployment of this model for live use.