Machine_Learning_Thesis_FER

June 8, 2023

Student Name: Muhammad Ali

Student ID: 3059828

Course: Big Data Management & Analytics

Griffith College Dublin

---

1 Use of Transfer Learning for Facial Emotion Recognition

This report is written to ensure the reproducibility of results obtained during the research on **The Use of Transfer Learning for Facial Emotion Recognition.** Refer to the following table of contents for specific material.

2 Section 1: Setup

This section includes the installations and imports of the necessary libraries for the code to run as well as the functions designed to be used in the code.

2.1 1.1 Installations

Some libraries were not available in Google Colab so all the following libraries were installed to ensure proper working of the code.

[ ]:

```
# Installing all libraries
!pip install tensorflow
!pip install pandas
!pip install matplotlib
!pip install numpy
!pip install scikit-learn
!pip install keras
!pip install mtcnn
!pip install openai
```

2.2 1.2 Imports

This section imports all necessary libraries required to run the code.

[6]:

```
# imports setup
import os, random, glob, shutil, time, json, requests, openai, math, cv2,␣
 ↪zipfile
import pandas as pd
import numpy as np
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
from mtcnn import MTCNN
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.utils import plot_model
from keras.layers import activation, Dense, Conv2D, MaxPool2D, Flatten,␣
 ↪BatchNormalization, GlobalAveragePooling2D, Input, Dropout
from keras.optimizers import Adam, SGD
from keras.metrics import categorical_crossentropy
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.applications.inception_v3 import InceptionV3,␣
 ↪preprocess_input
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from keras.models import load_model
# Mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2.3 1.3 Preprocessing Functions

This section includes functions design to be used for rescaling images, pre-processing images, visualising charts and graphs as well as plotting images.

[ ]:

```
# Function to rescale images
def rescale(image):
    size = 128 # set size
    image_size = (size, size, 3)
    height, width, channels = image.shape # get the shape of image

    #--------------------------------------------#
    img_white = np.ones(image_size, np.uint8)*255 # create an empty white image␣
 ↪for padding
    #--------------------------------------------#
    aspect_ratio = height/width # get aspect ratio of image

    # If image height is bigger than width, add padding width wise
    if aspect_ratio > 1:
        k = size/height
        wCal = math.ceil(k*width)
        img_resize = cv2.resize(image, (wCal, size))
        imgResizeShape = img_resize.shape
        wGap = math.ceil((size-wCal)/2)
        img_white[:, wGap:wGap+wCal] = img_resize
    # If image width is bigger than height, add padding height wise
    else:
        k = size/width
        hCal = math.ceil(k*height)
        img_resize = cv2.resize(image, (size, hCal))
        imgResizeShape = img_resize.shape
        hGap = math.ceil((size-hCal)/2)
        img_white[hGap:hGap+hCal, :] = img_resize
    return img_white
```

[ ]:

```
def count_files(file_dict):
    # Get the number of files for each key and calculate the total number of␣
 ↪training files
    file_counts = {key: len(paths) for key, paths in file_dict.items()}
    total_files = sum(file_counts.values())

    print(f'Number of Files by Each Class')
    print("----------------------------------------")
    # Output the number of files for each key
    for key, count in file_counts.items():
        print(f"{count}:\t{key}")

    # Output the separator line
    print("----------------------------------------")

    # Output the total number of training files
    print(f"Total Number of Training Files: {total_files}")
```

[ ]:

```
def plot_dictionary(dic):
    # Get the number of files for each key
    file_counts = [len(paths) for paths in dic.values()]

    # Get the keys as labels for the x-axis
    keys = list(dic.keys())

    # Create the bar chart
    plt.bar(keys, file_counts)

    # Customize the chart
    plt.xlabel('Keys')
    plt.ylabel('Number of Files')
    plt.title('File Count for Each Key')

    # Rotate the x-axis labels for better visibility (optional)
    plt.xticks(rotation=45)

    # Display the chart
    plt.show()
```

[ ]:

```
detector = MTCNN() # initialise face detector

# Function to preprocess images
def preprocess_image(img):
    global detector
    faces = detector.detect_faces(img) # detect faces
    if len(faces) > 0:
        # If a face is detected, crop the image to include only the face
        x, y, width, height = faces[0]['box'] #------------------------- take␣
 ↪coordinates of edges of face
        cropped_image = img[y:y+height, x:x+width] #-------------------- crop␣
 ↪the face using coordinates
        rescaled_image = rescale(cropped_image) #----------------------- set␣
 ↪aspect ratio using the function set previously
        gray_image = cv2.cvtColor(rescaled_image, cv2.COLOR_BGR2GRAY) #-␣
 ↪convert to gray scale
        img_eq = cv2.equalizeHist(gray_image) #-------------------------␣
 ↪perform gray scale normalization using histogram equalization
        return img_eq
    else:
        return None
```

[ ]:

```
# Function to plot image grid
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 10, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

[ ]:

```
# Takes history of model fitness and plots line charts for training and␣
 ↪validation loss as well as accuracy scores.
def plot_performance(history):
  # Plot the training and validation loss
  plt.plot(history.history['loss'], label='Training Loss')
  plt.plot(history.history['val_loss'], label='Validation Loss')
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()

  # Plot the training and validation accuracy
  plt.plot(history.history['accuracy'], label='Training Accuracy')
  plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.show()
```

3 Section 2: Data Pre-Processing

This section contains code blocks used to pre-process data. All datasets go through the same process but require different code structure to extract files from zips and arranging them accordingly but the high level process is same for all.

Following steps were taken for preprocessing of the above datasets in this notebook:

- Extracting datasets from Zip files
- Creating dictionaries with emotion classes as keys and list of image paths as values
- Use path dictionaries to read image files
- Use Preprocess function to detect, crop, add padding, convert to grayscale and normalise images
- Remove files that don't have faces
- Create directories for each emotion class
- Move pre-processed images into their respective class folders
- Delete original dataset folder once pre-processing is done and all files are copied to pre-processed folders. This is done to save space and avoid originals getting mixed with pre-processed images.

[ ]:

```
# Set Path Variables
DATASET = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets'#␣
 ↪Path to dataset folder
ZIPS = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/ZIPS'# Path to␣
 ↪zips folder
datasets = ['KDEF_and_AKDEF', 'JAFFE', 'Custom', 'openai_images']
emotions = ['anger','disgust','fear','happiness','neutral','sadness','surprise']

# Set Dataset directory paths for all datasets
KDEF = os.path.join(DATASET, datasets[0])
JAFFE = os.path.join(DATASET, datasets[1])
CUSTOM = os.path.join(DATASET, datasets[2])
OPENAI = os.path.join(DATASET, datasets[3])
```

```
for data in datasets:
    if not os.path.isdir(os.path.join(DATASET, data)):
        os.makedirs(os.path.join(DATASET, data))
        # Make class folders for custom dataset
        for emotion in emotions:
            if not os.path.isdir(os.path.join(DATASET, data, emotion)):
                os.mkdir(os.path.join(DATASET, data, emotion))
```

3.0.1 Note: Read Before Conintue

Next section is for data pre-processing. Steps involved in pre-processing take alot of time (**The running blocks were left over night in some cases**) essentially because of **large image sizes** and the need of **computational resources required** to go through several steps involved in pre-processing all the datasets.

Note: All the datasets have been extracted and these blocks does not need to be run.

In case of running the blocks, the dataset zips will be extracted that will change the structure of the dataset folders and the process will need to finish till the end to set the structure of dataset folder as well as causing additional time delays for moving to next steps as they cannot be run unless this process finishes.

In order to skip to the next runnable block to coninue testing the notebook with already pre-processed data, click here.

The datasets contained in the study are as follows:

- Custom Dataset (Mixed from following datasets)
- Karolinska Directed Emotional Faces - KDEF
- Japanese Female Facial Expression - JAFFE
- Images generated by OpenAI API - OpenAI Images

3.1 2.1 Generate Images Using Open AI DALL-E 2

```
[ ]: # Set output path for images and create directory if it doesnot exist
output_path = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪OpenAI_images'
if os.path.isdir(output_path) == False:
    os.mkdir(output_path)

'''
These are all Open AI API keys requested from my contacts for free use.
Apparently Open AI blocks the IP if abuse is detected.
Thats why not all keys were used.
--------------------------------------------------------------------------------
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 - used
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 - used
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 - used
```

```
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 - used
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 - used
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 - used
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 - Expired
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 - Expired
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 - Blocked
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 - Blocked
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 - Blocked
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 - Blocked
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 - Blocked
'''

# Authenticate with the OpenAI API using your API key
openai.api_key = "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"
```

[ ]:

```
# Define the text description for the desired facial expression
text = "A person with a surprised expression on their face with white␣
 ↪background"
```

[ ]:

```
# Set the parameters for the DALL-E 2 image generation
# REFERENCE: https://platform.openai.com/docs/guides/images/usage
count = 2000
for i in range(80):
    response = openai.Image.create(
      prompt=text, # Prompt to follow
      n=5, # Number of images to generate
      size="512x512" # Image size
    )
    # Get url of generated image from response to write the image on system
    for x in response['data']:
        print(x['url'])
        img = requests.get(x['url'])
        with open(f"{output_path}/surprise_expression_{count}.jpg", "wb") as f:
            f.write(img.content)
        count += 1
    time.sleep(58)
```

[ ]:

```
#----------------------------- Only for debug -------------------------------- #
count = 1
for x in response['data']:
    print(x['url'])
    img = requests.get(x['url'])
    with open(f"{output_path}/happiness_expression_{count}.jpg", "wb") as f:
        f.write(img.content)
    count += 1
```

3.2 2.2 Preprocess OpenAi Images Dataset

[ ]:

```
skipped_images = [] # List to contain images where MTCNN doesnot detect faces␣
 ↪so we can inspect them later.
if not os.path.isdir('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Datasets/Skipped_Images'):
  os.mkdir('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
 ↪Skipped_Images')
```

[ ]:

```
# Extract images from openai_images zip
openai_images = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/ZIPS/
 ↪openai_images.zip'
output = OPENAI
with zipfile.ZipFile(openai_images, 'r') as zip_ref:
    # extract all the files to the specified directory
    zip_ref.extractall(output)
```

[ ]:

```
# Create Dictionaries
folder = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
 ↪openai_images/openai_images'
# Create a dictionary and seperate image path for each emotion class using keys␣
 ↪as classes and values as lists containing paths.
openai_dic = {x : [os.path.join(folder, file) for file in os.listdir(folder) if␣
 ↪file.split('_')[0] == x] for x in emotions}
```

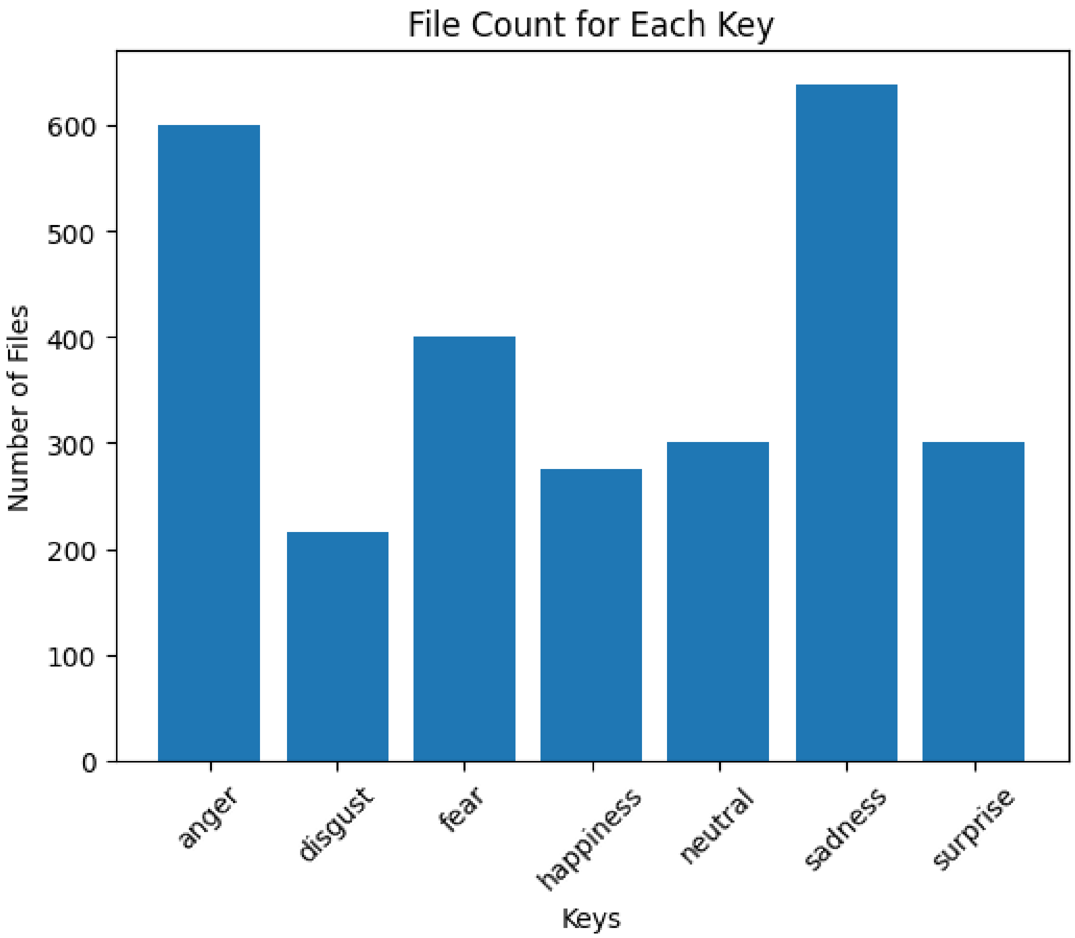
[ ]:

```
count_files(openai_dic) # Count files for Open ai directory
```

```
Number of Files by Each Class
----------------------------------------
600:    anger
216:    disgust
400:    fear
275:    happiness
301:    neutral
638:    sadness
300:    surprise
----------------------------------------
Total Number of Training Files: 2730
```

[ ]:

```
plot_dictionary(openai_dic) # Plot number of files
```



```
[ ]: # Show image samples for open ai images
     images_arr = []
     openai_folder = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
       ↪Datasets/openai_images/openai_images'
     for x in random.sample(glob.glob(f'{openai_folder}/*.[jJ][pP][gG]'), 10):
         image = cv2.imread(x)
         image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
         images_arr.append(image_rgb)
     plotImages(images_arr)
```

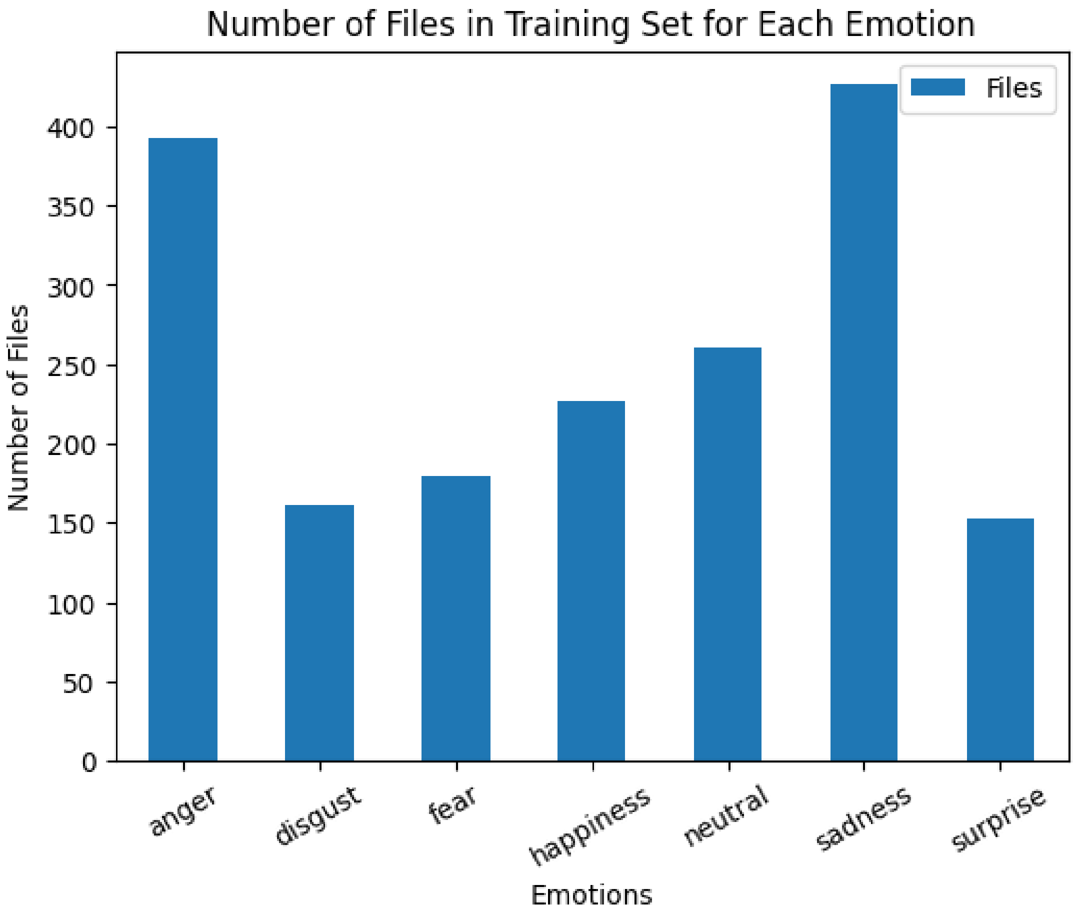
```
[ ]: # for each file path present in the list for each keys, read images and␣
      ↪preprocess them and write them in new folder.
     start = time.time()
     for key, value in openai_dic.items():
       # For each image read image, preprocess and write in new folder
       for path in value:
         clean_filepath = os.path.join(OPENAI, key, f'{path.split("/")[-1]}') #--␣
      ↪set path for output image
         if not os.path.isfile(clean_filepath):
           img = cv2.imread(path) #-----------------------------------------------␣
      ↪Read image
           img_eq = preprocess_image(img) #---------------------------------------␣
      ↪Detect, crop, convert to gray scale, normalize
           if img_eq is not None:
             cv2.imwrite(clean_filepath, img_eq) #--------------------------------␣
      ↪output the image
           else:
             shutil.copy(path, f'/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Datasets/Skipped_Images/{path.split("/
      ↪")[-1]}') # if face is not detected,move to this directory
         else:
           continue
     end = time.time()
     print(f'Time taken to extract: {end-start:.2f} seconds')
```

```
[ ]: # Count number files obtained after preprocessing
     count = 0
     list_OPENAI = []
     OPENAI_emotions = []
     for folder in emotions:
         #for file in os.listdir(os.path.join(KDEF_dir, folder)):
             num_files = len(os.listdir(os.path.join(OPENAI, folder)))
             count += num_files
             temp_list = [folder, num_files]
             list_OPENAI.append(temp_list)
             OPENAI_emotions.append(folder)
             print(f'Number of files for {folder.upper()}:\t{num_files}')
     print('-' * 40)
     print(f'Total Number of Training Files:\t{count}')
     print('-' * 40)
```

```
Number of files for ANGER:      393
Number of files for DISGUST:    161
Number of files for FEAR:       180
Number of files for HAPPINESS:  227
Number of files for NEUTRAL:    261
Number of files for SADNESS:    426
Number of files for SURPRISE:   153
---------------------------------------
Total Number of Training Files: 1801
---------------------------------------
```

```
[ ]: # plot number of files after pre-processing
     df_openai = pd.DataFrame(list_OPENAI, columns=['Emotions', 'Files'],
                          index=OPENAI_emotions)

     df_openai.plot(kind='bar')

     plt.xticks(rotation=30, horizontalalignment="center")
     plt.title('Number of Files in Training Set for Each Emotion')
     plt.xlabel('Emotions')
     plt.ylabel('Number of Files')
```

```
[ ]: Text(0, 0.5, 'Number of Files')
```



[ ]:

```
# ----------------------------Be Careful ---------------------------------------#
# Remove original folder since we have obtained preprocessed images.
shutil.rmtree('/content/drive/MyDrive/FER/datasets/openai_images/openai_images')
```

3.3 2.3 Preprocess KDEF Images Dataset

[ ]:

```
# Extract KDEF dataset zip
kdef_zip = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/ZIPS/
 ↪KDEF_and_AKDEF.zip'
kdef_output = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
 ↪KDEF_and_AKDEF'
if not os.path.isdir(kdef_output):
    os.mkdir(kdef_output)
with zipfile.ZipFile(kdef_zip, 'r') as zip_ref:
    # extract all the files to the specified directory
    zip_ref.extractall(kdef_output)
```

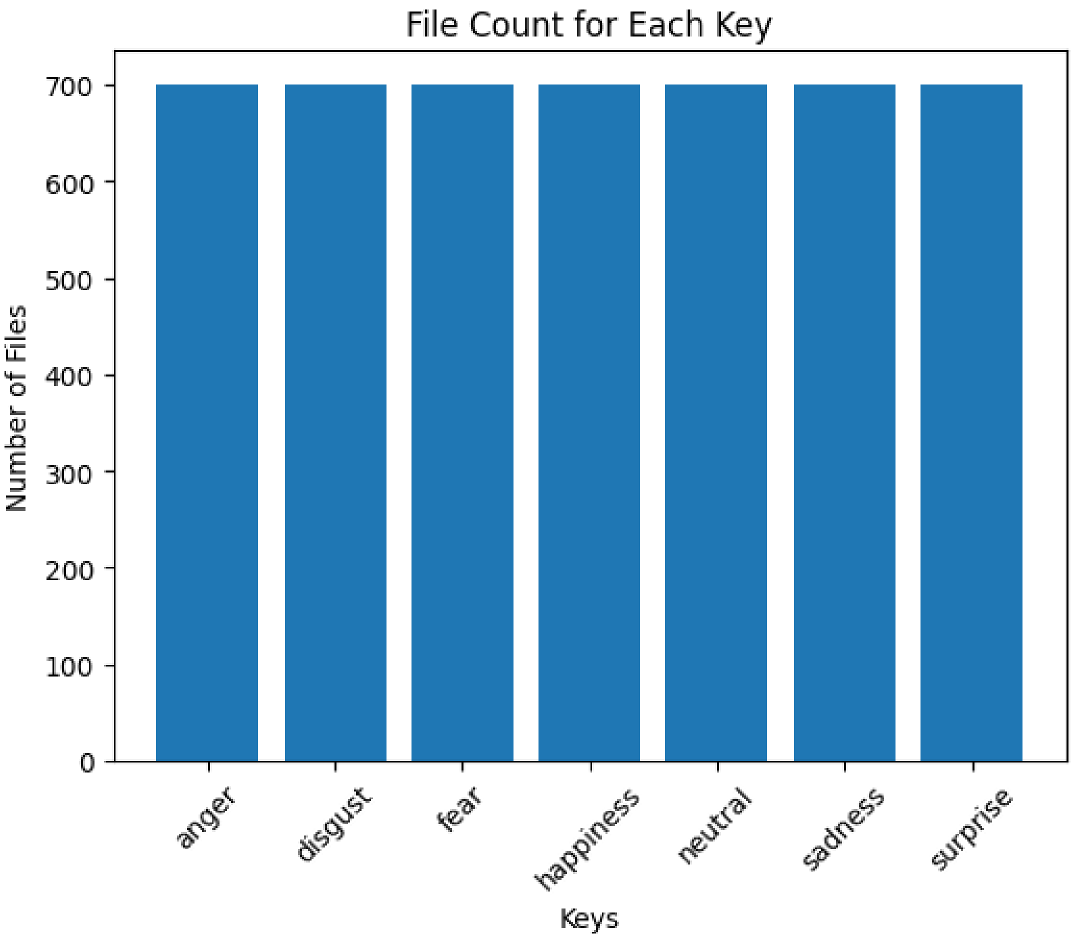
[ ]:

```
# Create image path dictionaries
kdef_emotions = ['AF','AN','DI','HA','NE','SA','SU']
kdef_extract = os.path.join(KDEF, 'KDEF_and_AKDEF','KDEF')
kdef_dic = {x : [] for x in emotions}
for folder in os.listdir(kdef_extract):
  for file in os.listdir(os.path.join(kdef_extract, folder)):
    filepath = os.path.join(kdef_extract, folder, file)
    letter = file[4:6]
    if letter == 'AF':
        #fear
        kdef_dic['fear'].append(filepath)
    elif letter == 'AN':
        #anger
        kdef_dic['anger'].append(filepath)
    elif letter == 'DI':
        #disgusted
        kdef_dic['disgust'].append(filepath)
    elif letter == 'HA':
        #happiness
        kdef_dic['happiness'].append(filepath)
    elif letter == 'NE':
        #neutral
        kdef_dic['neutral'].append(filepath)
    elif letter == 'SA':
        #sadness
        kdef_dic['sadness'].append(filepath)
    elif letter == 'SU':
        #surprise
        kdef_dic['surprise'].append(filepath)
```

```
[ ]: count_files(kdef_dic) # count files
```

```
Number of Files by Each Class
----------------------------------------
700:    anger
700:    disgust
700:    fear
700:    happiness
700:    neutral
699:    sadness
699:    surprise
----------------------------------------
Total Number of Training Files: 4898
```

```
[ ]: plot_dictionary(kdef_dic) # plot files
```



[ ]:

```
# show sample images of KDEF
images_arr = []
openai_folder = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Datasets/KDEF_and_AKDEF/KDEF_and_AKDEF/KDEF/AF01'
for x in random.sample(glob.glob(f'{openai_folder}/*.[jJ][pP][gG]'), 10):
    image = cv2.imread(x)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    images_arr.append(image_rgb)
plotImages(images_arr)
```

[ ]:

```
skipped_images = []
count = 0
# Perform the processing similar to before
start = time.time()
for key, value in kdef_dic.items():
  # For each image read image, preprocess and write in new folder
  for path in value:
    clean_filepath = os.path.join(KDEF, key, f'{path.split("/")[-1]}') #-- set␣
 ↪path for output image
    if os.path.isfile(clean_filepath) == False and os.path.isfile(f'/content/
 ↪drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/Skipped_Images/{path.
 ↪split("/")[-1]}') == False:
      try:
        img = cv2.imread(path) # read images
        img_eq = preprocess_image(img) # detect, crop, convert to grayscale,␣
 ↪normalise
        if img_eq is not None:
          cv2.imwrite(clean_filepath, img_eq) #---------------------------␣
 ↪output the image
        else:
          shutil.copy(path, f'/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Datasets/Skipped_Images/{path.split("/
 ↪")[-1]}') # if face is not detected,move to this directory
      except:
        count += 1
        pass
    else:
      print(clean_filepath)
end = time.time()
```

```
print(f'Time taken to extract: {end-start:.2f} seconds')
print(count)
```

[ ]:

```
# Create path dictionaries for AKDEF dataset
akdef_emotions = ['AF','AN','DI','HA','NE','SA','SU']
akdef_extract = os.path.join(KDEF, 'KDEF_and_AKDEF','AKDEF')
akdef_dic = {x : [] for x in emotions}
for file in os.listdir(akdef_extract):
    filepath = os.path.join(akdef_extract, file)
    letter = file[1:3]
    if letter == 'AF':
        #fear
        akdef_dic['fear'].append(filepath)
    elif letter == 'AN':
        #anger
        akdef_dic['anger'].append(filepath)
    elif letter == 'DI':
        #disgusted
        akdef_dic['disgust'].append(filepath)
    elif letter == 'HA':
        #happiness
        akdef_dic['happiness'].append(filepath)
    elif letter == 'NE':
        #neutral
        akdef_dic['neutral'].append(filepath)
    elif letter == 'SA':
        #sadness
        akdef_dic['sadness'].append(filepath)
    elif letter == 'SU':
        #surprise
        akdef_dic['surprise'].append(filepath)
```

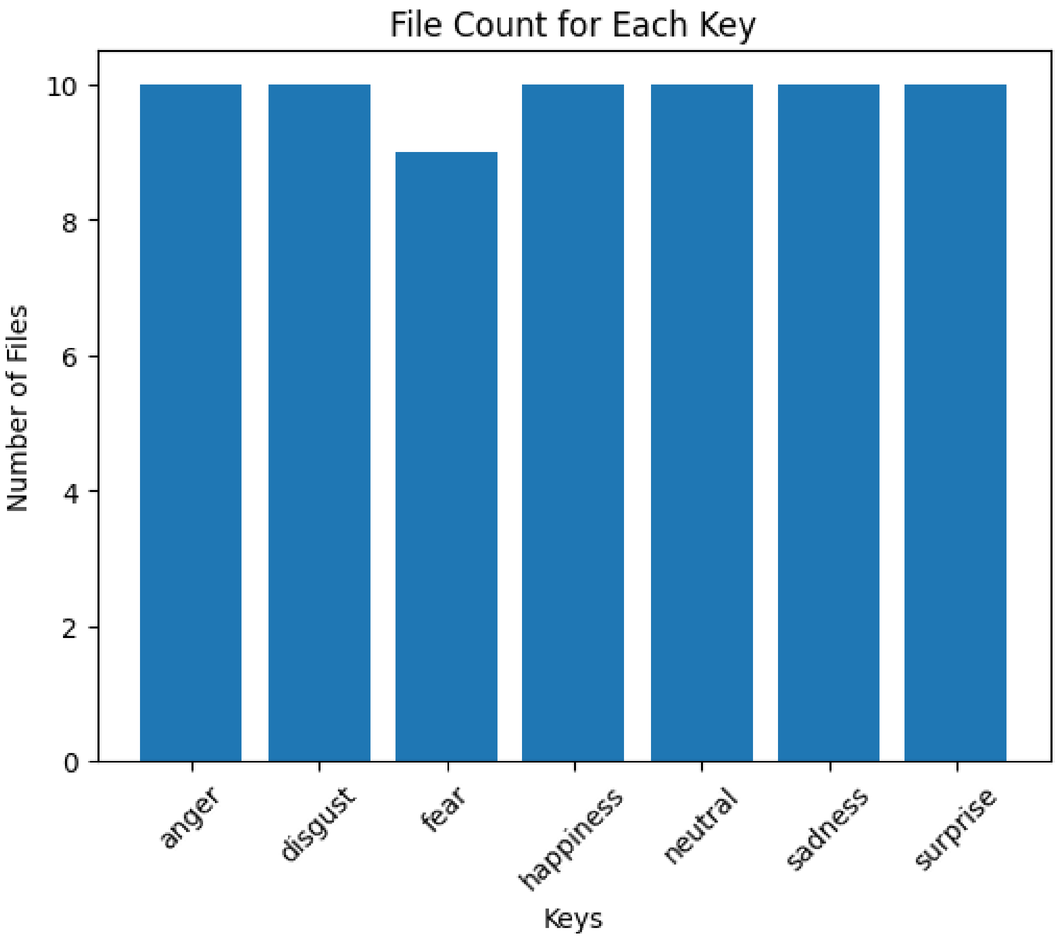
[ ]:

```
count_files(akdef_dic) # count files
```

```
Number of Files by Each Class
----------------------------------------
10:     anger
10:     disgust
9:      fear
10:     happiness
10:     neutral
10:     sadness
10:     surprise
----------------------------------------
Total Number of Training Files: 69
```

[ ]:

```
plot_dictionary(akdef_dic) # plot files
```



[ ]:
```
skipped_images = []
# Preprocess the dataset
start = time.time()
for key, value in akdef_dic.items():
  # For each image read image, preprocess and write in new folder
  for path in value:
    clean_filepath = os.path.join(KDEF, key, f'{path.split("/")[-1]}') #-- set␣
 ↪path for output image
    if not os.path.isfile(clean_filepath):
      img = cv2.imread(path) # read images
      img_eq = preprocess_image(img) # detect, crop, grayscale, normalize
      if img_eq is not None:
        cv2.imwrite(clean_filepath, img_eq) #--------------------------- output␣
 ↪the image
      else:
        shutil.copy(path, f'/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Datasets/Skipped_Images/{path.split("/
 ↪")[-1]}') # if face is not detected,move to this directory
    else:
      continue
end = time.time()
print(f'Time taken to extract: {end-start:.2f} seconds')
```

[ ]:

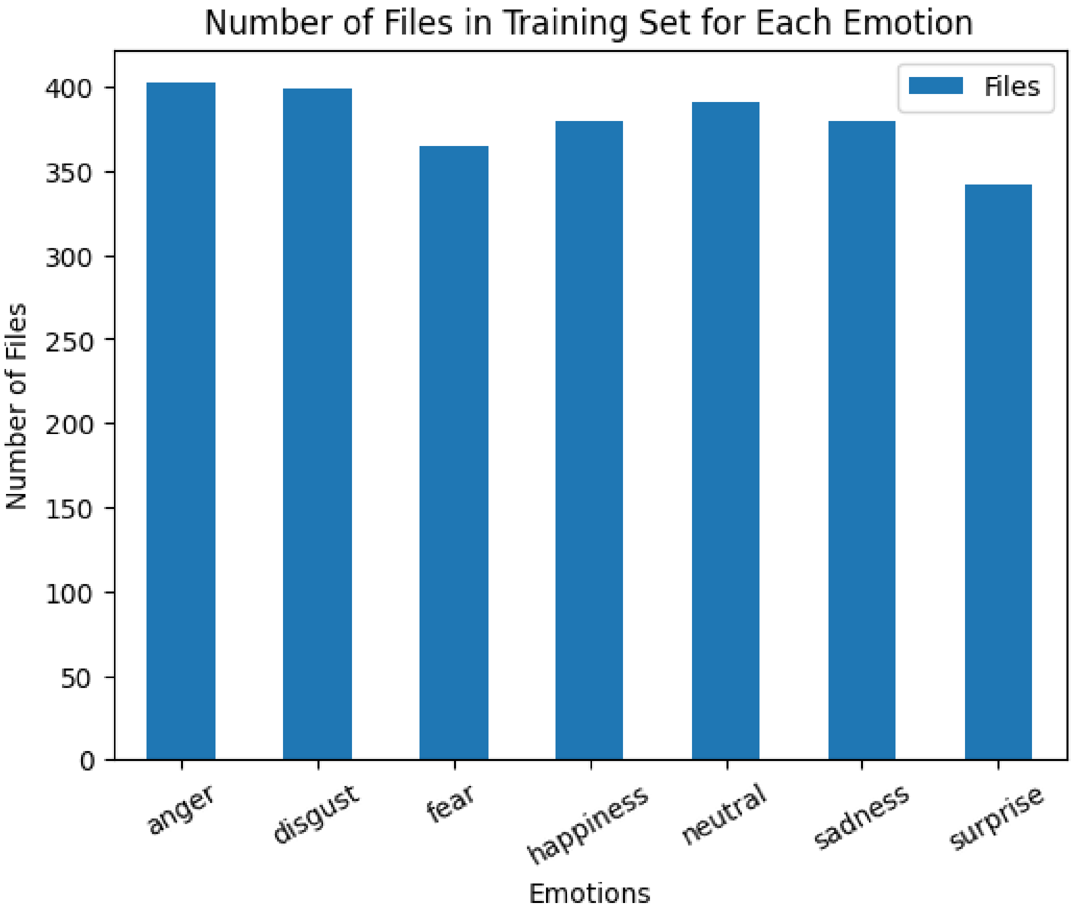
```
# Count number files obtained after preprocessing
count = 0
list_KDEF = []
KDEF_emotions = []
for folder in emotions:
    #for file in os.listdir(os.path.join(KDEF_dir, folder)):
        num_files = len(os.listdir(os.path.join(KDEF, folder)))
        count += num_files
        temp_list = [folder, num_files]
        list_KDEF.append(temp_list)
        KDEF_emotions.append(folder)
        print(f'Number of files for {folder.upper()}:\t{num_files}')
print('-' * 40)
print(f'Total Number of Training Files:\t{count}')
print('-' * 40)
```

```
Number of files for ANGER:      402
Number of files for DISGUST:    399
Number of files for FEAR:       365
Number of files for HAPPINESS:  379
Number of files for NEUTRAL:    391
Number of files for SADNESS:    379
Number of files for SURPRISE:   342
----------------------------------------
Total Number of Training Files: 2657
----------------------------------------
```

[ ]:

```
# plot number of files after pre-processing
df_kdef = pd.DataFrame(list_KDEF, columns=['Emotions', 'Files'],
                  index=KDEF_emotions)

df_kdef.plot(kind='bar')

plt.xticks(rotation=30, horizontalalignment="center")
plt.title('Number of Files in Training Set for Each Emotion')
plt.xlabel('Emotions')
plt.ylabel('Number of Files')
```

```
[ ]: Text(0, 0.5, 'Number of Files')
```



[ ]:
```
# ----------------------------Be Careful ---------------------------------------#
# Remove original folder since we have obtained preprocessed images.
shutil.rmtree('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
  ↪KDEF_and_AKDEF/KDEF_and_AKDEF')
```

3.4 2.4 Prepropcess JAFFE Images Dataset

[ ]:
```
# Extract Jaffe dataset zip
jaffe_zip = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/ZIPS/JAFFE.
  ↪zip'
jaffe_output = JAFFE
if not os.path.isdir(jaffe_output):
    os.mkdir(jaffe_output)
with zipfile.ZipFile(jaffe_zip, 'r') as zip_ref:
    # extract all the files to the specified directory
    zip_ref.extractall(jaffe_output)
```

[ ]:

```
# Organize directories to get all images in one folder.
if os.path.isdir(f'{JAFFE}/test1'):
    print('Directories Already Organized')
else:
    for folder in os.listdir(f'{JAFFE}/JAFFE'):
        shutil.move(f'{JAFFE}/JAFFE/' + folder, f'{JAFFE}')
    for x in (os.listdir(f'{JAFFE}/test1')):
        filename = x.split('.')[0] + '.' + x.split('.')[1] + 't.' + x.split('.
 ↪')[2]
        shutil.move(os.path.join(f'{JAFFE}/test1', x), os.path.
 ↪join(f'{JAFFE}','train', filename))
    os.rmdir(f'{JAFFE}/JAFFE')
    os.rmdir(f'{JAFFE}/test1')
    print('Organized Directories Successfully')
```

```
Organized Directories Successfully
```

[ ]:

```
# Create path dictionary for images
jaffe_train = os.path.join(JAFFE, 'train')
jaffe_dic = {x : [os.path.join(jaffe_train, file) for file in os.
 ↪listdir(jaffe_train) if x in file and 'jpg' in file] for x in emotions}
```

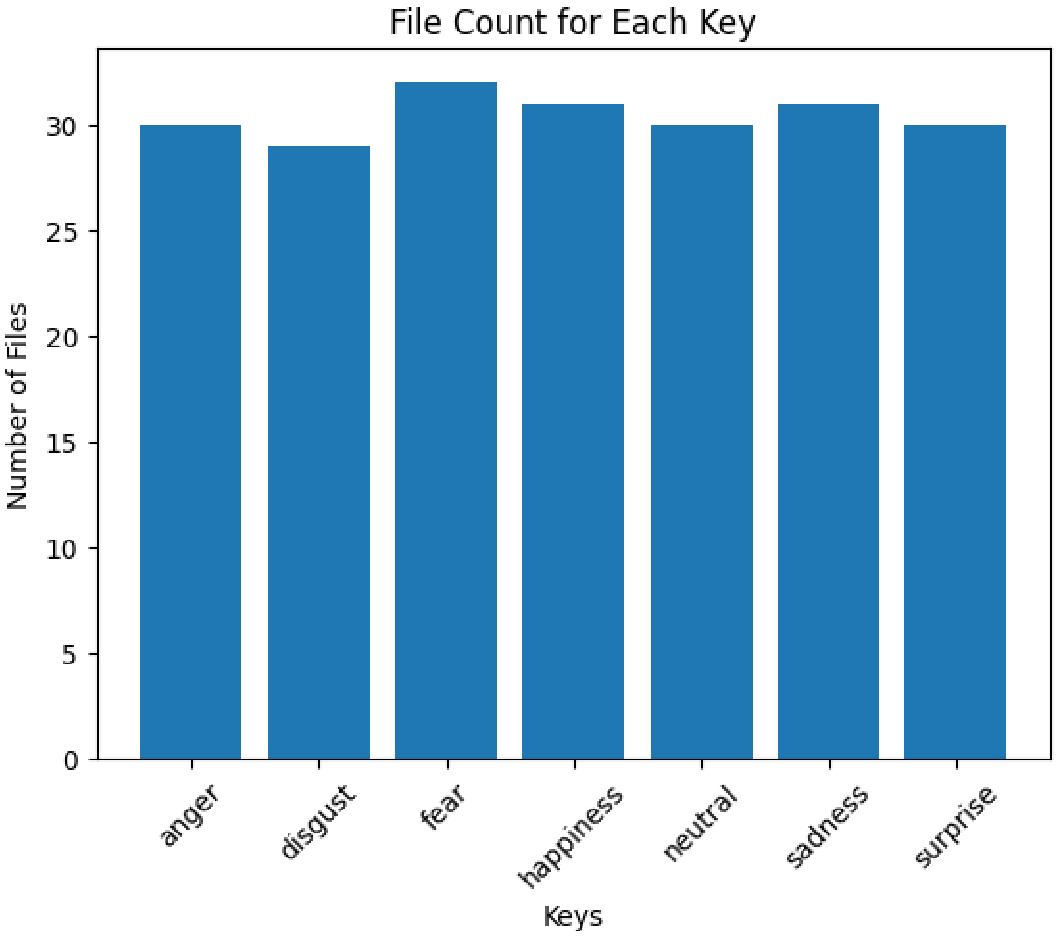
[ ]:

```
count_files(jaffe_dic) # Count Files
```

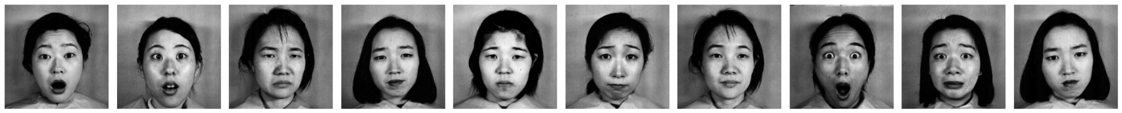
```
Number of Files by Each Class
----------------------------------------
30:     anger
29:     disgust
32:     fear
31:     happiness
30:     neutral
31:     sadness
30:     surprise
----------------------------------------
Total Number of Training Files: 213
```

[ ]:

```
plot_dictionary(jaffe_dic) # Plot Files
```



```
[ ]: # Show sample images for JAFFE
images_arr = []
openai_folder = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Datasets/JAFFE/train'
for x in random.sample(glob.glob(f'{openai_folder}/*.[jJ][pP][gG]'), 10):
    image = cv2.imread(x)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    images_arr.append(image_rgb)
plotImages(images_arr)
```



[ ]:

```
skipped_images = []
# Preprocess dataset
start = time.time()
for key, value in jaffe_dic.items():
  # For each image read image, preprocess and write in new folder
  for path in value:
    clean_filepath = os.path.join(JAFFE, key, f'{path.split("/")[-1]}') #-- set␣
 ↪path for output image
    if not os.path.isfile(clean_filepath):
      img = cv2.imread(path) #read image
      img_eq = preprocess_image(img)# detect, crop, grascale, normalize
      if img_eq is not None:
        cv2.imwrite(clean_filepath, img_eq) #--------------------------- output␣
 ↪the image
      else:
        shutil.copy(path, f'/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Datasets/Skipped_Images/{path.split("/
 ↪")[-1]}') # if face is not detected,move to this directory
    else:
      continue
end = time.time()
print(f'Time taken to extract: {end-start:.2f} seconds')
```

[ ]:

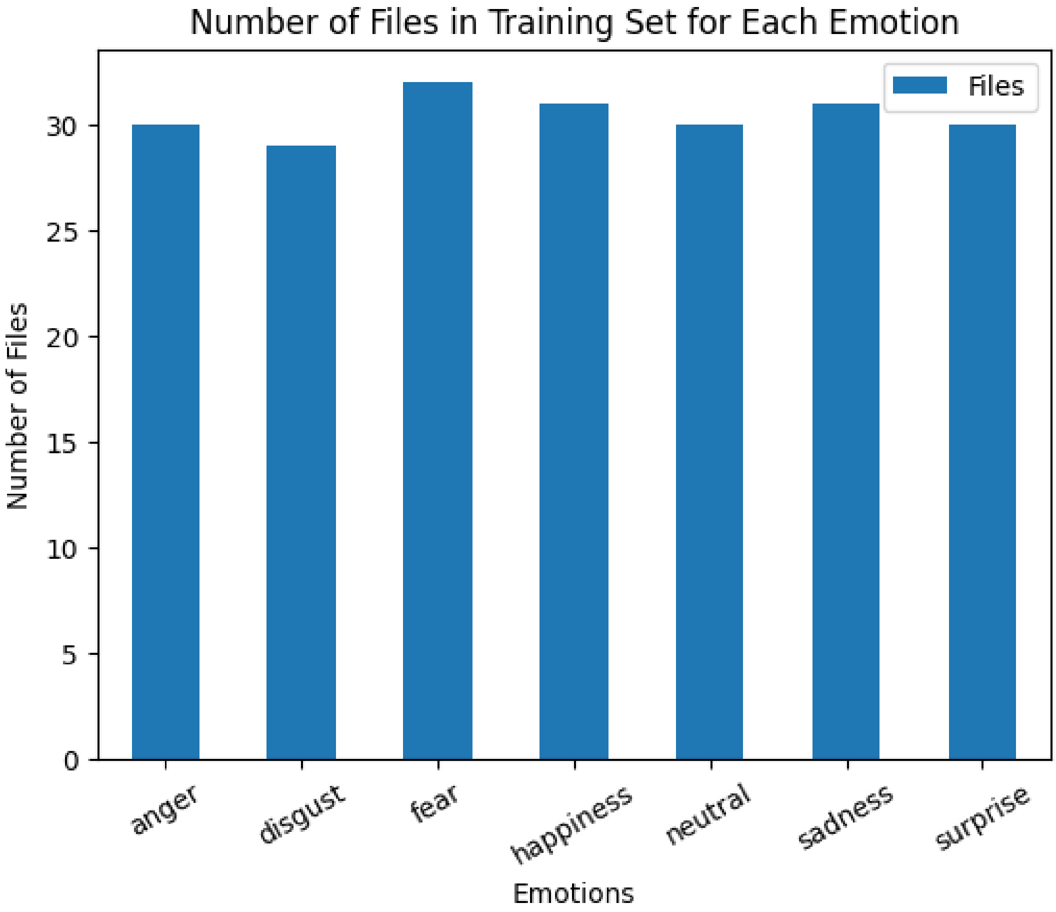
```
# Count number of files
count = 0
list_JAFFE = []
JAFFE_emotions = []
for folder in emotions:
    #for file in os.listdir(os.path.join(KDEF_dir, folder)):
        num_files = len(os.listdir(os.path.join(JAFFE, folder)))
        count += num_files
        temp_list = [folder, num_files]
        list_JAFFE.append(temp_list)
        JAFFE_emotions.append(folder)
        print(f'Number of files for {folder.upper()}:\t{num_files}')
print('-' * 40)
print(f'Total Number of Training Files:\t{count}')
print('-' * 40)
```

```
Number of files for ANGER:      30
Number of files for DISGUST:    29
Number of files for FEAR:       32
Number of files for HAPPINESS:  31
Number of files for NEUTRAL:    30
Number of files for SADNESS:    31
Number of files for SURPRISE:   30
----------------------------------------
```

```
Total Number of Training Files: 213
----------------------------------------
```

[ ]:
```
# Plot number of files after pre-processing
df_jaffe = pd.DataFrame(list_JAFFE, columns=['Emotions', 'Files'],
                  index=JAFFE_emotions)

df_jaffe.plot(kind='bar')

plt.xticks(rotation=30, horizontalalignment="center")
plt.title('Number of Files in Training Set for Each Emotion')
plt.xlabel('Emotions')
plt.ylabel('Number of Files')
```

[ ]:
```
Text(0, 0.5, 'Number of Files')
```

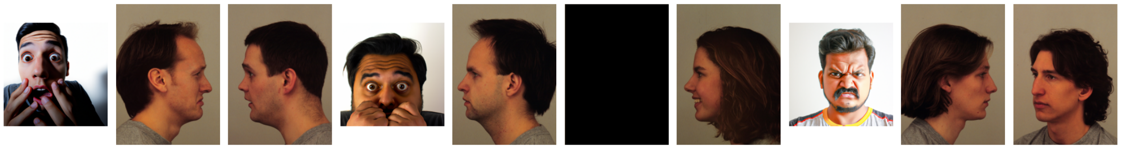


[ ]:
```
# ----------------------------Be Careful -------------------------------------#
#Remove original dataset folder as we have preprocessed images stored
```

```
shutil.rmtree('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
 ↪JAFFE/train')
```

3.5 2.5 Skipped Images

```
[ ]: # Show images that were skipped during pre-processing
     images_arr = []
     openai_folder = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Datasets/Skipped_Images'
     for x in random.sample(glob.glob(f'{openai_folder}/*.[jJ][pP][gG]'), 10):
         image = cv2.imread(x)
         image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
         images_arr.append(image_rgb)
     plotImages(images_arr)
```



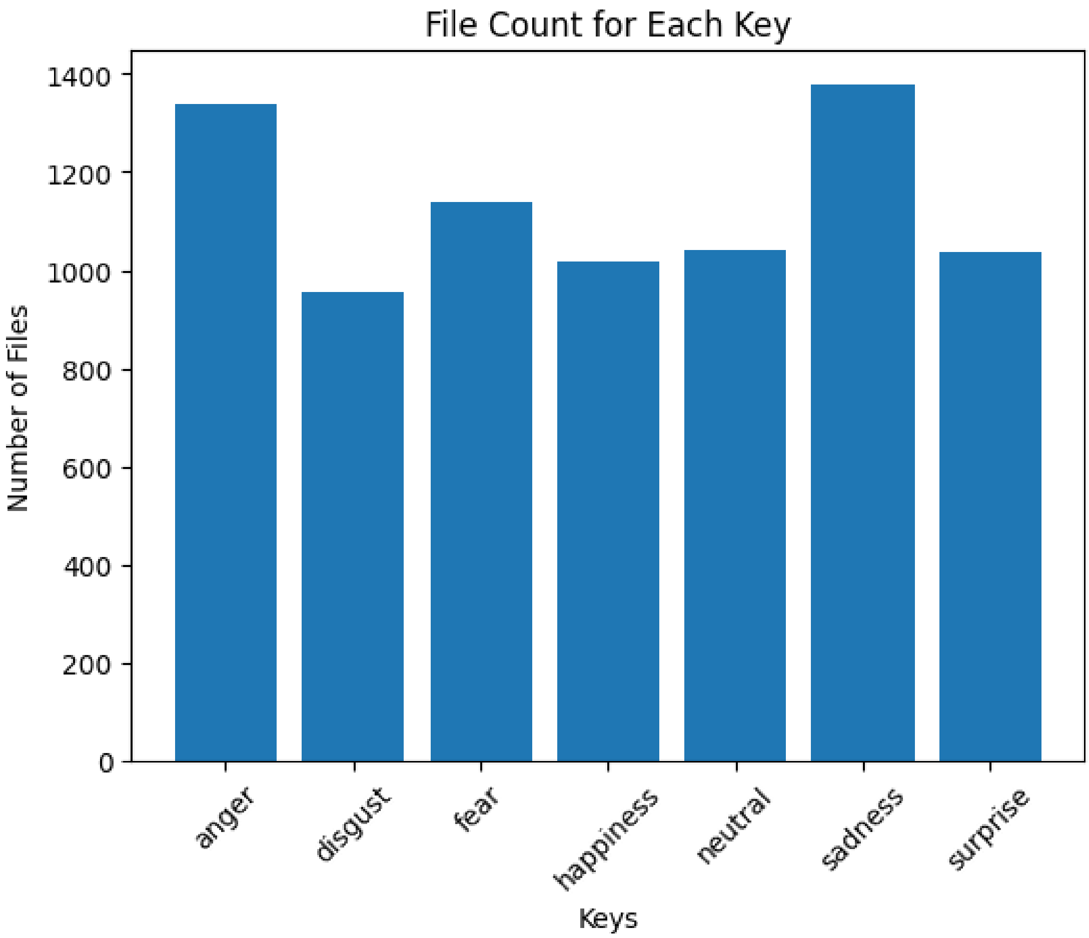
3.6 2.6 Create Master Dataset

This section creates a custom dataset by mixing all files preprocessed from previous datasets.

```
[ ]: from collections import defaultdict

     # List of dictionaries
     dictionaries = [openai_dic, kdef_dic, akdef_dic, jaffe_dic]

     # Create a defaultdict with lists as default values
     merged_dict = defaultdict(list)

     # Merge the dictionaries
     for dictionary in dictionaries:
         for key, value in dictionary.items():
             merged_dict[key].extend(value)

     # Convert the merged_dict back to a regular dictionary
     merged_dict = dict(merged_dict)

     count_files(merged_dict)
     plot_dictionary(merged_dict)
```

```
Number of Files by Each Class
----------------------------------------
```

```
1340:   anger
955:    disgust
1141:   fear
1016:   happiness
1041:   neutral
1378:   sadness
1039:   surprise
----------------------------------------
Total Number of Training Files: 7910
```



[ ]:

```
# Copy files from all dataset folders to custom dataset folder
folders = [JAFFE, KDEF, OPENAI]
for folder in folders:
  for emotion in emotions:
    for file in os.listdir(os.path.join(folder, emotion)):
      filepath = os.path.join(folder, emotion, file)
      output_path = os.path.join(CUSTOM, emotion, file)
      shutil.copy(filepath, output_path)
```

[ ]:

```
# Count number of files obtained after copy
count = 0
list_custom = []
custom_emotions = []
for folder in emotions:
    #for file in os.listdir(os.path.join(KDEF_dir, folder)):
        num_files = len(os.listdir(os.path.join(CUSTOM, folder)))
        count += num_files
        temp_list = [folder, num_files]
        list_custom.append(temp_list)
        custom_emotions.append(folder)
        print(f'Number of files for {folder.upper()}:\t{num_files}')
print('-' * 40)
print(f'Total Number of Training Files:\t{count}')
print('-' * 40)
```

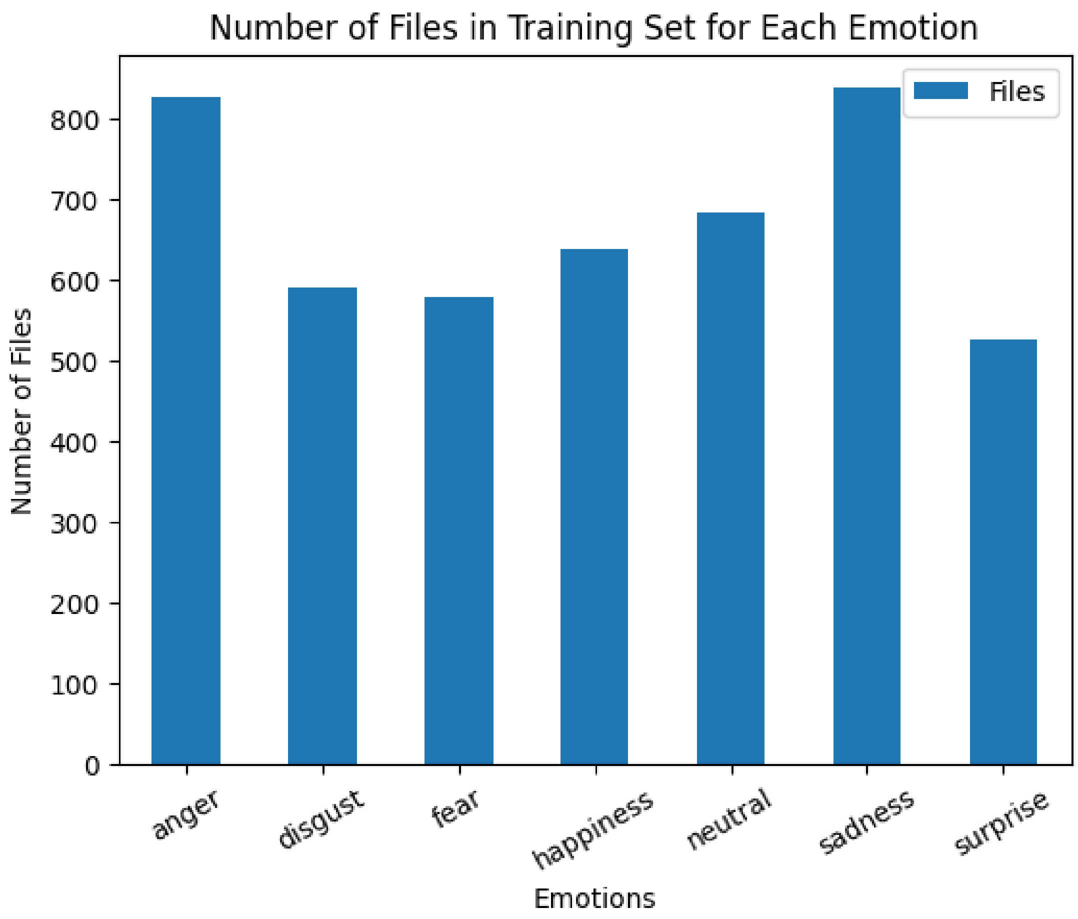
```
Number of files for ANGER:      825
Number of files for DISGUST:    589
Number of files for FEAR:       577
Number of files for HAPPINESS:  637
Number of files for NEUTRAL:    682
Number of files for SADNESS:    836
Number of files for SURPRISE:   525
----------------------------------------
Total Number of Training Files: 4671
----------------------------------------
```

[ ]:

```
df_custom = pd.DataFrame(list_custom, columns=['Emotions', 'Files'],
                  index=custom_emotions)

df_custom.plot(kind='bar')

plt.xticks(rotation=30, horizontalalignment="center")
plt.title('Number of Files in Training Set for Each Emotion')
plt.xlabel('Emotions')
plt.ylabel('Number of Files')
```

[ ]:

```
Text(0, 0.5, 'Number of Files')
```



3.7 2.7 Data Augmentation

[ ]:

```
# Copy the clean dataset
CUSTOM_COPY = CUSTOM + '_augmented'
#shutil.copytree(CUSTOM, CUSTOM_COPY)
```

[ ]:

```
# Create Directories for Training, Validation and Testing sets in KDEF folder
print('Creating directories for training, validation and testing...')
if os.path.isdir(CUSTOM_COPY + '/training') is False:
    for folder in os.listdir(CUSTOM_COPY):
        os.makedirs(f'{CUSTOM_COPY}/training/{folder}')
        os.makedirs(f'{CUSTOM_COPY}/validation/{folder}')
        os.makedirs(f'{CUSTOM_COPY}/testing/{folder}')
print('Moving Files...')
for folder in emotions:
    for x in random.sample(glob.glob(f'{CUSTOM_COPY}/{folder}/*.[jJ][pP][gG]'), 150):
        shutil.move(x, f'{CUSTOM_COPY}/testing/{folder}')
    for x in random.sample(glob.glob(f'{CUSTOM_COPY}/{folder}/*.[jJ][pP][gG]'),␣
  ↪150):
        shutil.move(x, f'{CUSTOM_COPY}/validation/{folder}')
    for x in random.sample(glob.glob(f'{CUSTOM_COPY}/{folder}/*.[jJ][pP][gG]'),␣
  ↪len(glob.glob(f'{CUSTOM_COPY}/{folder}/*.[jJ][pP][gG]'))):
        shutil.move(x, f'{CUSTOM_COPY}/training/{folder}')
print('Removing Empty Directories...')
# Remove Empty Directories
for folder in emotions:
    os.rmdir(f'{CUSTOM_COPY}/{folder}')
print('-'*20 + 'Done' + '-'*20)
```

```
Creating directories for training, validation and testing…
Moving Files…
Removing Empty Directories…
--------------------Done--------------------
```

[ ]:

```
testing = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
  ↪Custom_augmented/testing'
training = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
  ↪Custom_augmented/training'
validation = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
  ↪Custom_augmented/validation'
```

[ ]:

```
import os
import cv2
import re

# Define the directory containing your emotion class folders
data_directory = training

# Get the list of emotion class folders
emotion_classes = os.listdir(data_directory)

# Loop through each emotion class folder
for emotion_class in emotion_classes:
    # Get the path of the emotion class folder
    emotion_class_path = os.path.join(data_directory, emotion_class)

    # Get the list of image files in the emotion class folder
    image_files = os.listdir(emotion_class_path)

    smallest_class_size = 225
    # Loop through each image file
    for image_file in image_files:
        # Get the path of the image file
        image_path = os.path.join(emotion_class_path, image_file)
        # Read the image
        image = cv2.imread(image_path)

        # Flip the image horizontally
        flipped_image = cv2.flip(image, 1)

        # Generate the filename for the flipped image
        flipped_image_file = "flipped_" + image_file

        # Save the flipped image in the same directory
        flipped_image_path = os.path.join(emotion_class_path,␣
 ↪flipped_image_file)
        cv2.imwrite(flipped_image_path, flipped_image)

        # Decrement the number of flipped images to generate
        smallest_class_size -= 1

        # Break the loop if the desired number of flipped images has been␣
 ↪reached
        if smallest_class_size == 0:
            break
```

[ ]:

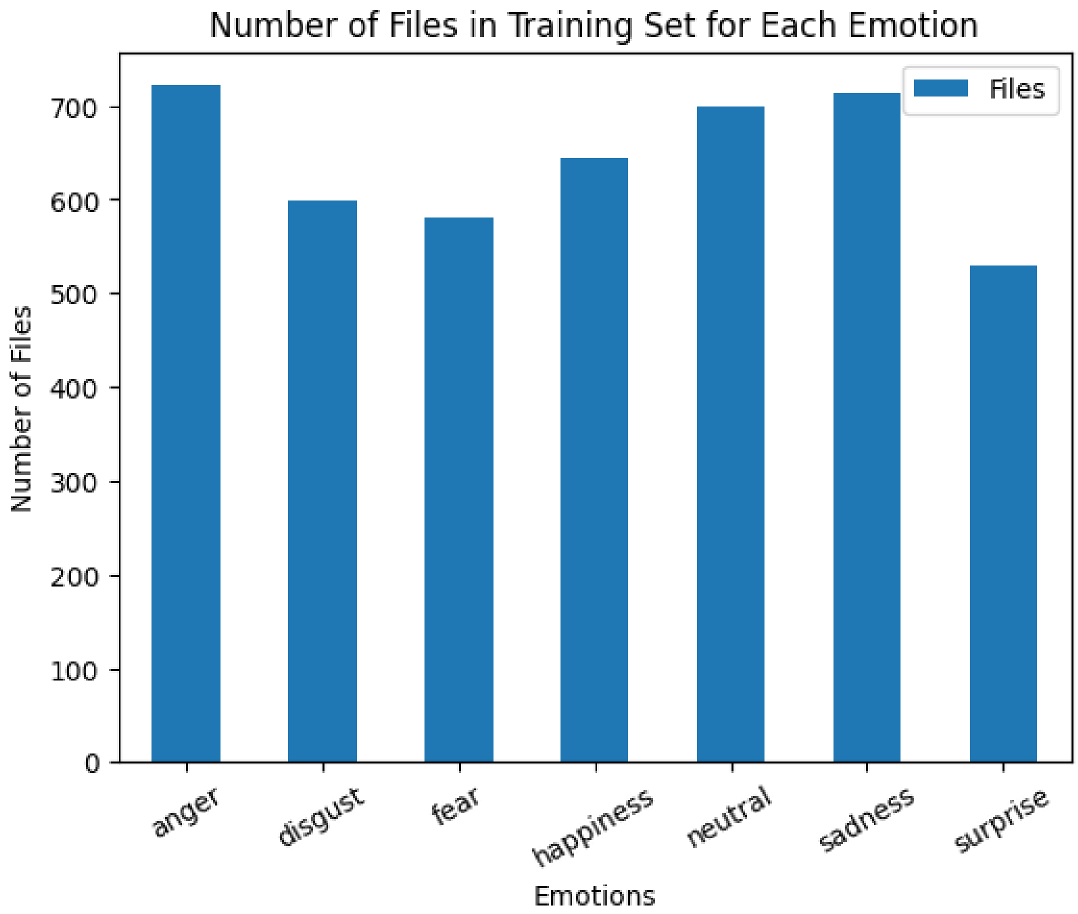
```
# Count number of files obtained after copy
count = 0
list_custom_ab = []
custom_ab_emotions = []
for folder in emotions:
    #for file in os.listdir(os.path.join(KDEF_dir, folder)):
        num_files = len(os.listdir(os.path.join(training, folder)))
        count += num_files
        temp_list = [folder, num_files]
        list_custom_ab.append(temp_list)
        custom_ab_emotions.append(folder)
        print(f'Number of files for {folder.upper()}:\t{num_files}')
print('-' * 40)
print(f'Total Number of Training Files:\t{count}')
print('-' * 40)
```

```
Number of files for ANGER:      721
Number of files for DISGUST:    598
Number of files for FEAR:       580
Number of files for HAPPINESS:  643
Number of files for NEUTRAL:    700
Number of files for SADNESS:    713
Number of files for SURPRISE:   530
----------------------------------------
```

```
Total Number of Training Files: 4485
----------------------------------------
```

```
[ ]: df_custom_ab = pd.DataFrame(list_custom_ab, columns=['Emotions', 'Files'],
                       index=custom_ab_emotions)

     df_custom_ab.plot(kind='bar')

     plt.xticks(rotation=30, horizontalalignment="center")
     plt.title('Number of Files in Training Set for Each Emotion')
     plt.xlabel('Emotions')
     plt.ylabel('Number of Files')
```

```
[ ]: Text(0, 0.5, 'Number of Files')
```



```
[ ]: CLEAN_DS = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/
       ↪Dataset'
     CUSTOM_COPY = CUSTOM + '_augmented'
```

```
#shutil.copytree(CLEAN_DS, CUSTOM_COPY)
```

[ ]:

```
'/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Datasets/Custom_augmente
d'
```

4 Section 3: Machine Learning

This section includes the Machine Learning part.

Following three models are used for training data:

- Xception
- Inception V3
- Custom Made CNN

The section is subdivided into following three sections for easy navigation:

- Training Models
- Fine-Tuning Models
- Testing Models

[ ]:

```
# Check the number fo files in dataset folder
# Note: Each training, validation and testing folder appears seven times because there are seven class folders in each set for emotions.
CUSTOM_COPY = CUSTOM + '_augmented'
CleanDS_Copy = CUSTOM_COPY
count = 0
list_custom = []
custom_emotions = []
for folder in os.listdir(CleanDS_Copy):
    for file in os.listdir(os.path.join(CleanDS_Copy, folder)):
        num_files = len(os.listdir(os.path.join(CleanDS_Copy, folder, file)))
        count += num_files
        temp_list = [folder, num_files]
        list_custom.append(temp_list)
        custom_emotions.append(folder)
        print(f'Number of files for {folder.upper()}:\t{num_files}')
print('-' * 40)
print(f'Total Number of Training Files:\t{count}')
print('-' * 40)
```

[ ]:

```
# set paths for training, validation and testing directories
training = CUSTOM_COPY + '/training'
validation = CUSTOM_COPY + '/validation'
testing = CUSTOM_COPY + '/testing'
emotions = os.listdir(training)
```

[ ]:

```
# Creating batches of preprocessed images using ImageDataGenerator from Keras
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True
)
```

[ ]:

```
# Set batches for training, validation and testing sets
target_size = (128,128)
training_batches = train_datagen.flow_from_directory(directory=training,␣
 ↪target_size=target_size, classes=emotions, batch_size=64)
validation_batches = train_datagen.flow_from_directory(directory=validation,␣
 ↪target_size=target_size, classes=emotions, batch_size=64)
testing_batches = train_datagen.flow_from_directory(directory=testing,␣
 ↪target_size=target_size, classes=emotions, batch_size=64, shuffle=False)
```

```
Found 4485 images belonging to 7 classes.
Found 700 images belonging to 7 classes.
Found 700 images belonging to 7 classes.
```

[ ]:

```
# View sample input images
imgs, labels = next(training_batches)
```

[ ]:

```
plotImages(imgs)
```

4.1 3.1 Training Models

[ ]:

```
from keras.callbacks import ModelCheckpoint

if not os.path.isdir('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models'):
  os.mkdir('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/Models')

xception_model_path = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/xception.hdf5'
inception_model_path = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/inception.hdf5'
sequential_model_path = '/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/custom_sequence.hdf5'
```

4.1.1 Xception

[ ]:

```
#from tensorflow.keras import layers
base_model = Xception(weights='imagenet', include_top=False,␣
 ↪input_shape=(128,128,3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(7, activation='softmax')(x)

# Define the complete model
xception = Model(inputs=base_model.input, outputs=predictions)


# define the model checkpoint callback
X_checkpoint = ModelCheckpoint(xception_model_path,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 5s 0us/step
```

[ ]:

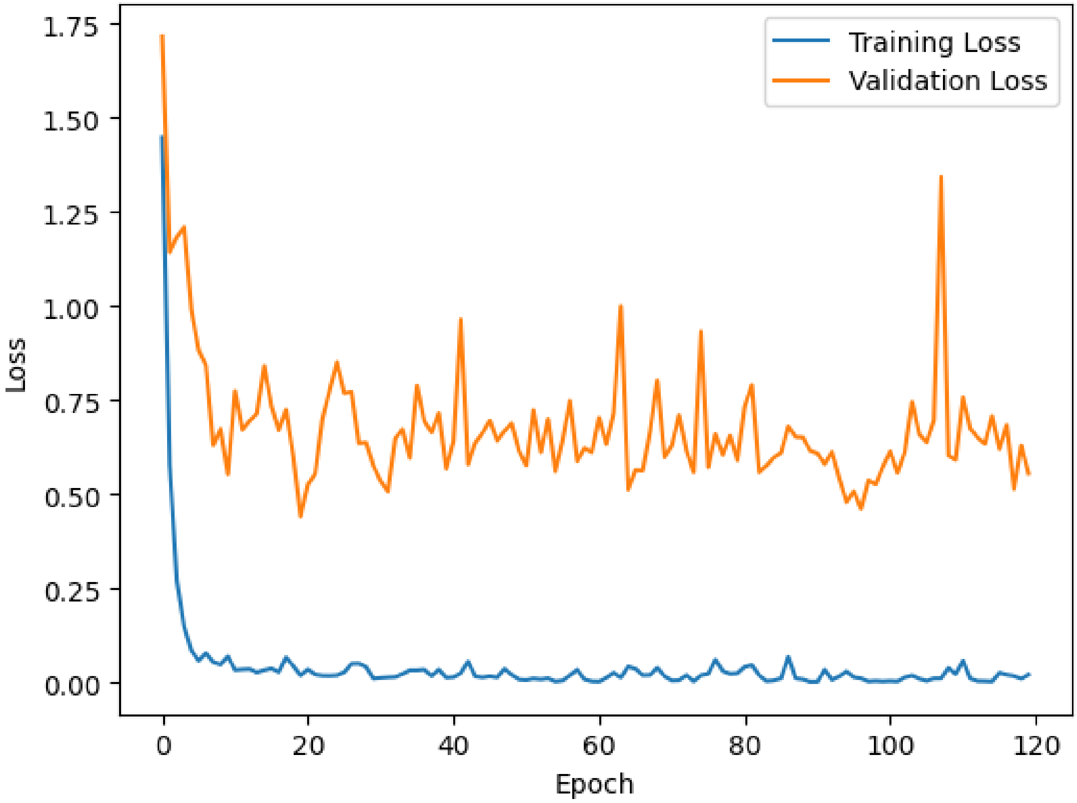
```
from tensorflow.keras.utils import plot_model

# Plot the model architecture
plot_model(xception, to_file='/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/xception_architecture.png', show_shapes=True)
```

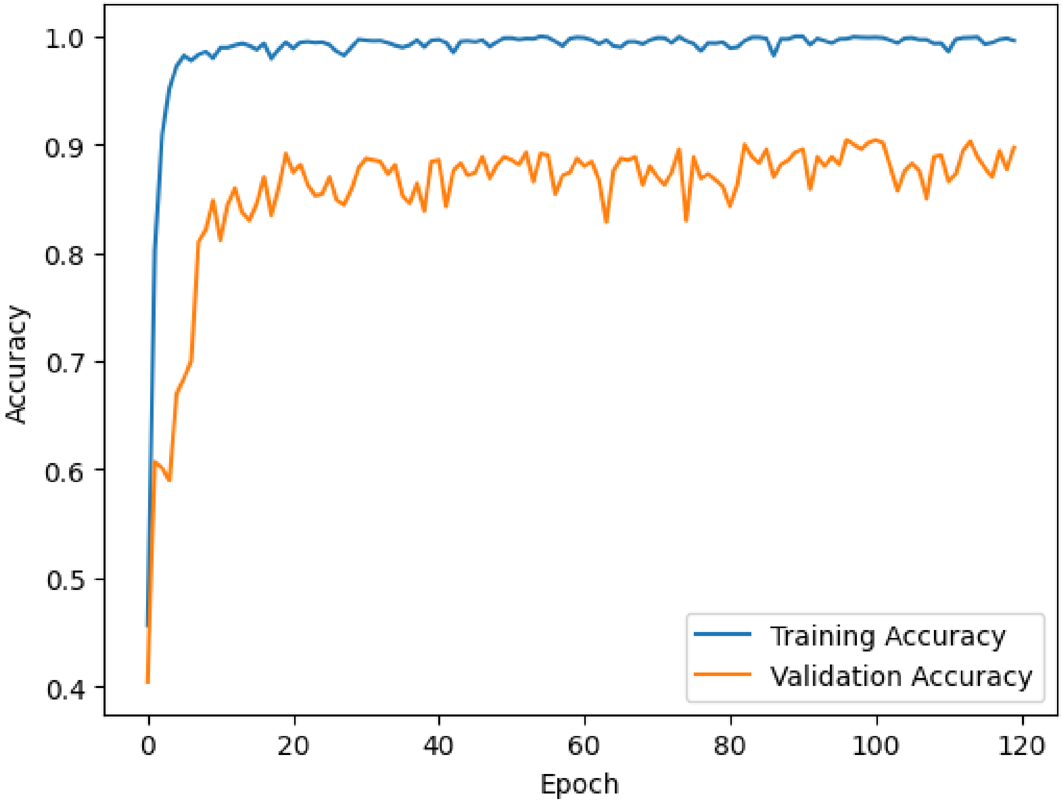
[ ]:

```
xception.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = xception.fit(x=training_batches, validation_data=validation_batches,␣
 ↪epochs=120, verbose=2, callbacks=[X_checkpoint])
```

[ ]:

```
# Plot the training and validation loss
plot_performance(history)
```


Training Loss
Validation Loss
Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
120
Epoch



4.1.2 Inception V3

[ ]:

```
# Define the input shape
img_width, img_height = 128, 128
input_shape = (img_width, img_height, 3)

# Load the pre-trained Inception model
base_model = InceptionV3(weights='imagenet', include_top=False,␣
 ↪input_shape=input_shape)

# Add a global average pooling layer and a dense layer for classification
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(7, activation='softmax')(x)

# Define the complete model
inception = Model(inputs=base_model.input, outputs=predictions)

# define the model checkpoint callback
```

```
I_checkpoint = ModelCheckpoint(inception_model_path,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 7s 0us/step
```

[ ]:

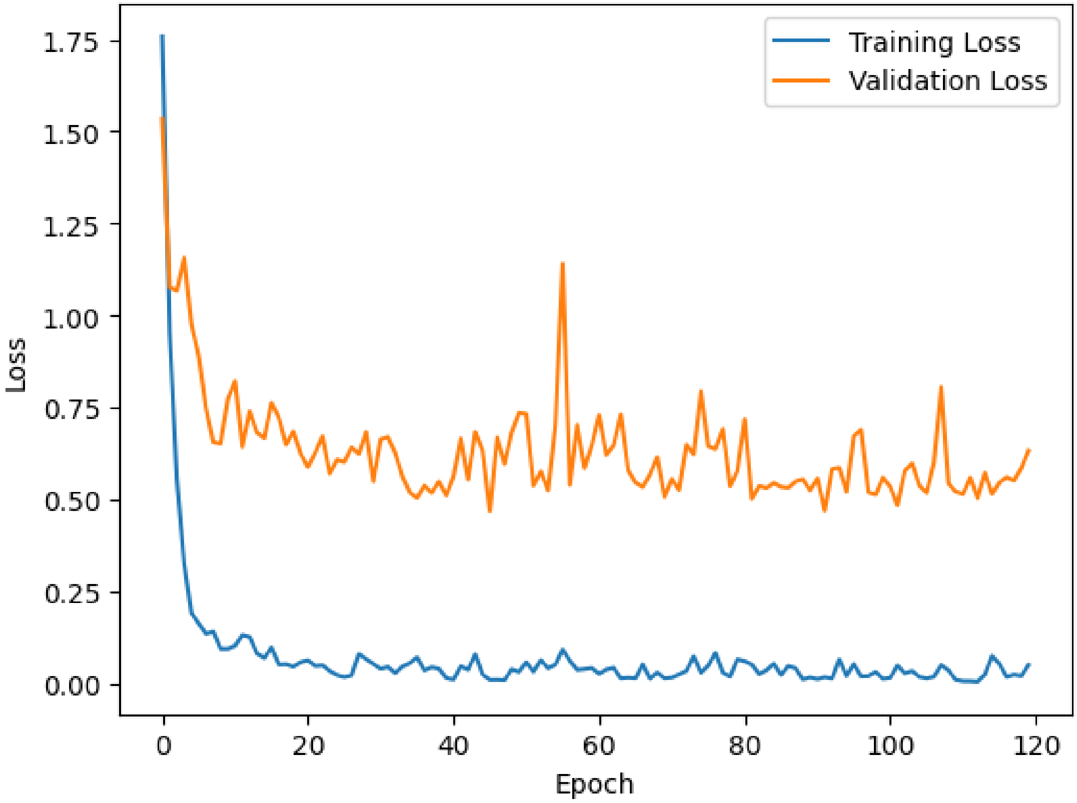
```
# Plot the model architecture
plot_model(inception, to_file='/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/inception_architecture.png',␣
 ↪show_shapes=True)
```

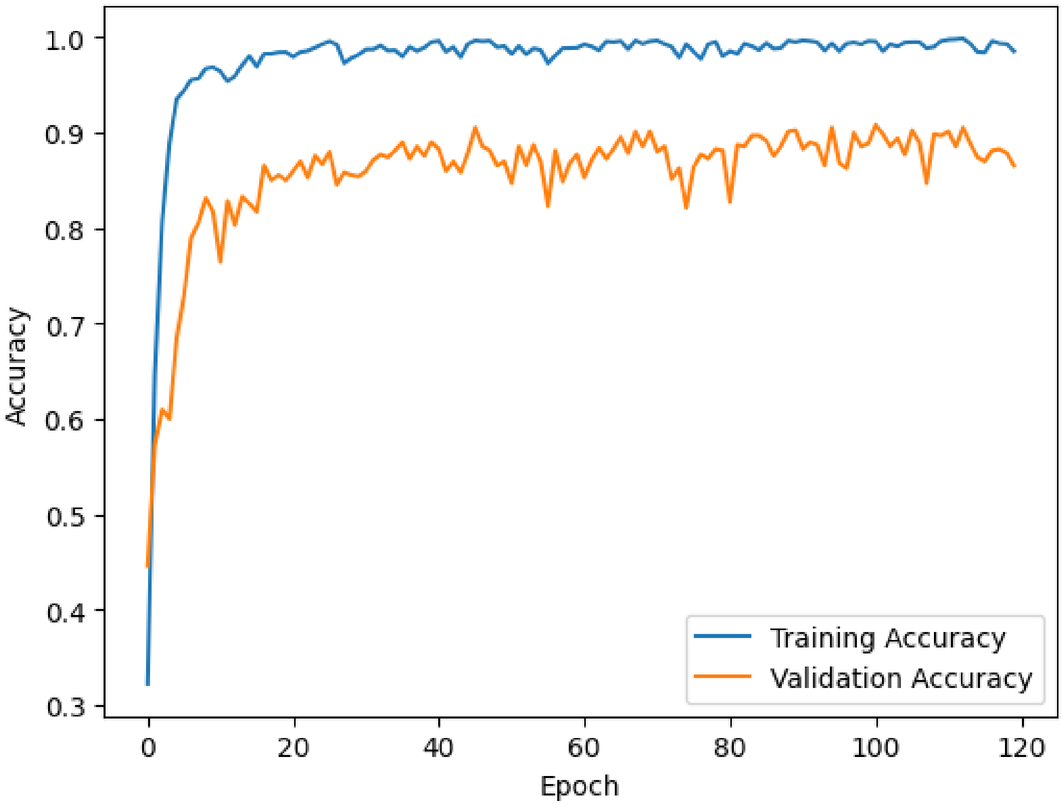
[ ]:

```
# Train the model
#history = xception.fit(train_data, epochs=epochs, validation_data=val_data)
inception.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
Inception_history_R = inception.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[I_checkpoint])
```

[ ]:

```
plot_performance(Inception_history_R)
```


Training Loss
Validation Loss
Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
120
Epoch



4.1.3 Sequential

[ ]:

```
from tensorflow.keras import layers

# Define the CNN model
model = tf.keras.Sequential()

# Convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128,␣
 ↪3)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
```

```python
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

# Flatten the feature maps
model.add(layers.Flatten())

# Fully connected layers
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(7, activation='softmax'))  # 7 classes for emotions

# define the model checkpoint callback
M_checkpoint = ModelCheckpoint(sequential_model_path,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 126, 126, 32)      896

 batch_normalization_4 (Batc  (None, 126, 126, 32)     128
 hNormalization)

 max_pooling2d_16 (MaxPoolin  (None, 63, 63, 32)       0
 g2D)

 conv2d_29 (Conv2D)          (None, 61, 61, 64)        18496

 batch_normalization_5 (Batc  (None, 61, 61, 64)       256
 hNormalization)

 max_pooling2d_17 (MaxPoolin  (None, 30, 30, 64)       0
 g2D)

 conv2d_30 (Conv2D)          (None, 28, 28, 128)       73856

 batch_normalization_6 (Batc  (None, 28, 28, 128)      512
 hNormalization)

 max_pooling2d_18 (MaxPoolin  (None, 14, 14, 128)      0
 g2D)

 conv2d_31 (Conv2D)          (None, 12, 12, 256)       295168

 batch_normalization_7 (Batc  (None, 12, 12, 256)      1024
 hNormalization)

 max_pooling2d_19 (MaxPoolin  (None, 6, 6, 256)        0
 g2D)

 flatten_5 (Flatten)         (None, 9216)              0

 dense_16 (Dense)            (None, 64)                589888

 dropout_1 (Dropout)         (None, 64)                0

 dense_17 (Dense)            (None, 7)                 455

=================================================================
Total params: 980,679
Trainable params: 979,719
Non-trainable params: 960
_________________________________________________________________
```

[ ]:

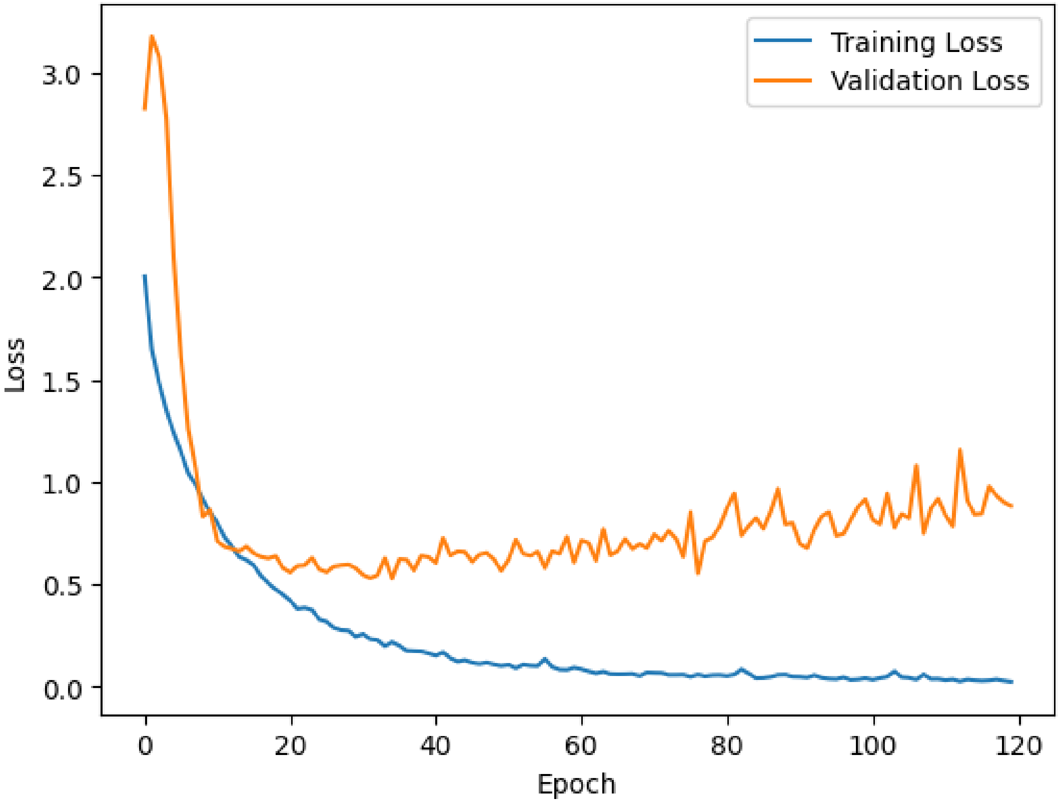
```
# Plot the model architecture
plot_model(model, to_file='/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/model_architecture.png', show_shapes=True)
```

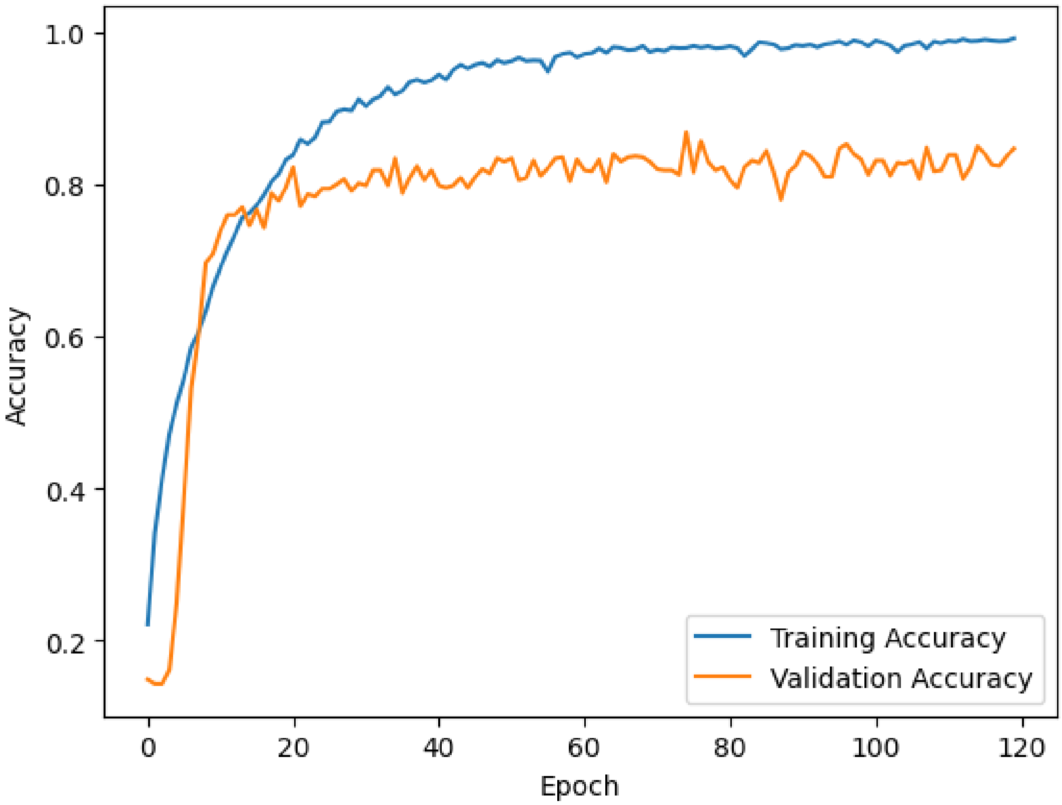
[ ]:

```
# Train the model
#history = xception.fit(train_data, epochs=epochs, validation_data=val_data)
model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
model_history_R = model.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[M_checkpoint])
```

[ ]:

```
plot_performance(model_history_R)
```


Training Loss
Validation Loss
Loss
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
120
Epoch



4.2 3.2 Fine-Tuning Models

4.2.1 Fine-Tuning Xception

Optimizer SGD

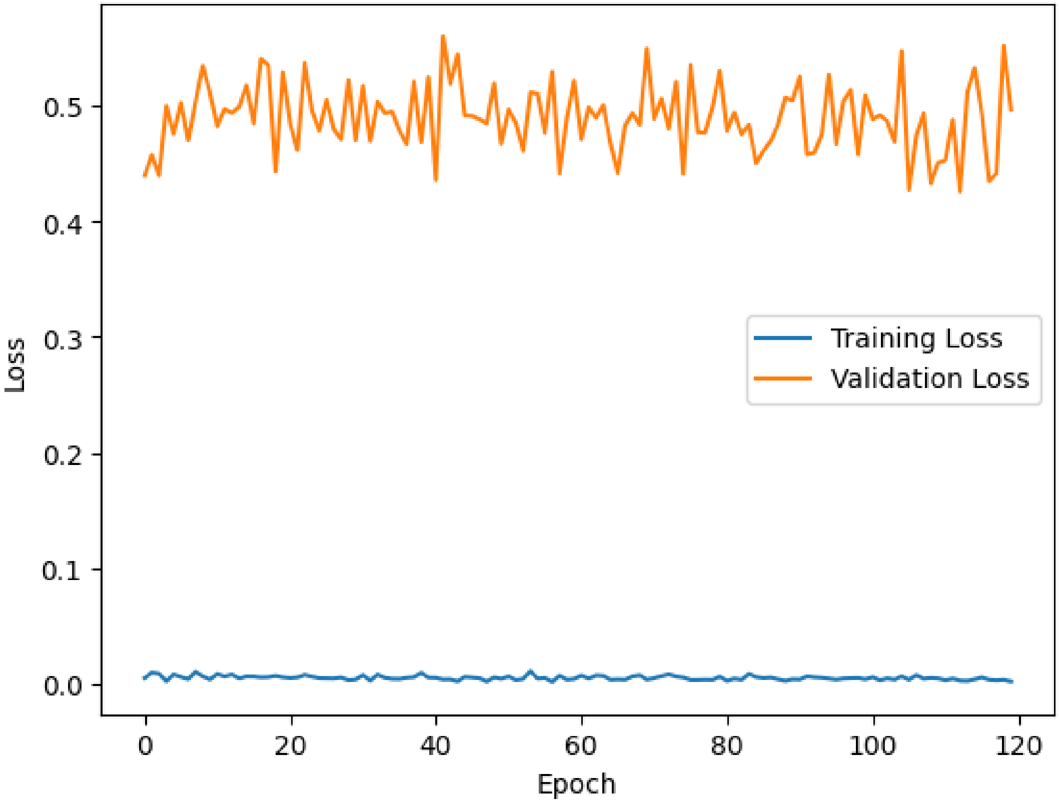
[ ]:
```
# define the model checkpoint callback
XFTSGDCP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTSGDCP.hdf5',
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

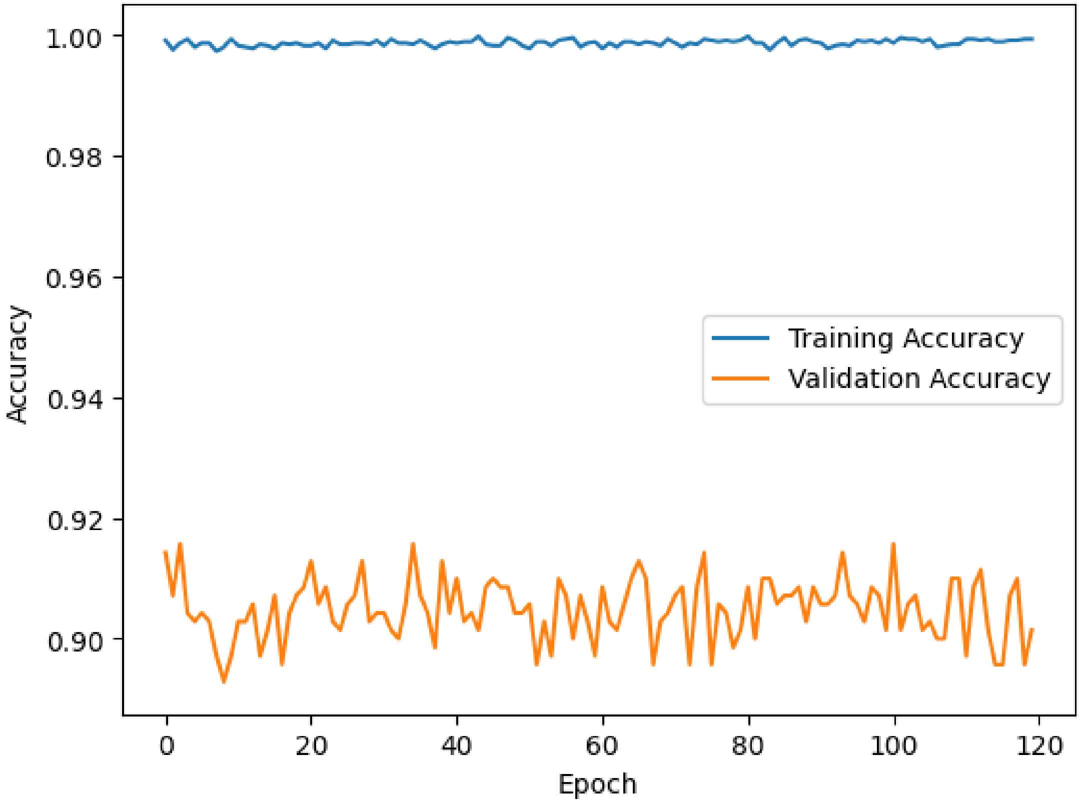
[ ]:
```
from keras.models import load_model

# Load the saved model
xception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/xception.hdf5')
```

[ ]:
```
xception.compile(optimizer=SGD(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
```

```
SGD_Histroy = xception.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[XFTSGDCP])
```

```
[ ]: plot_performance(SGD_Histroy)
```





Optimizer ADAGrade

[ ]:

```
# define the model checkpoint callback
XFTADAGRADECP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTADAGRADECP.hdf5',
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

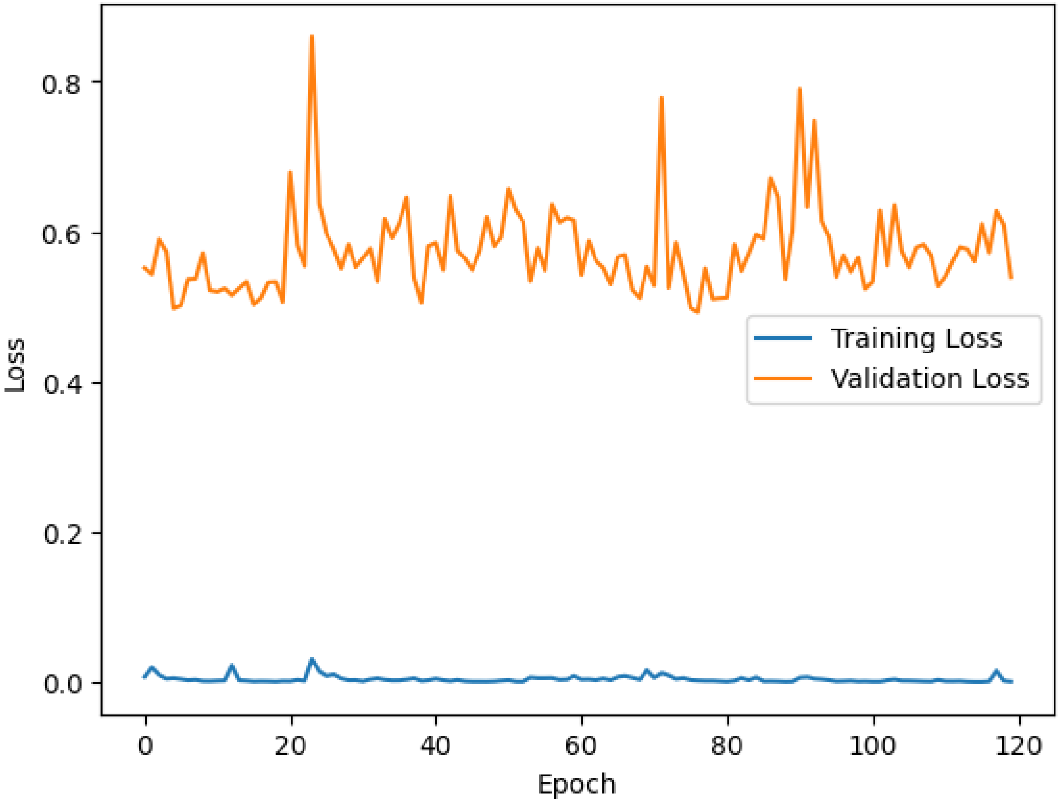
[ ]:

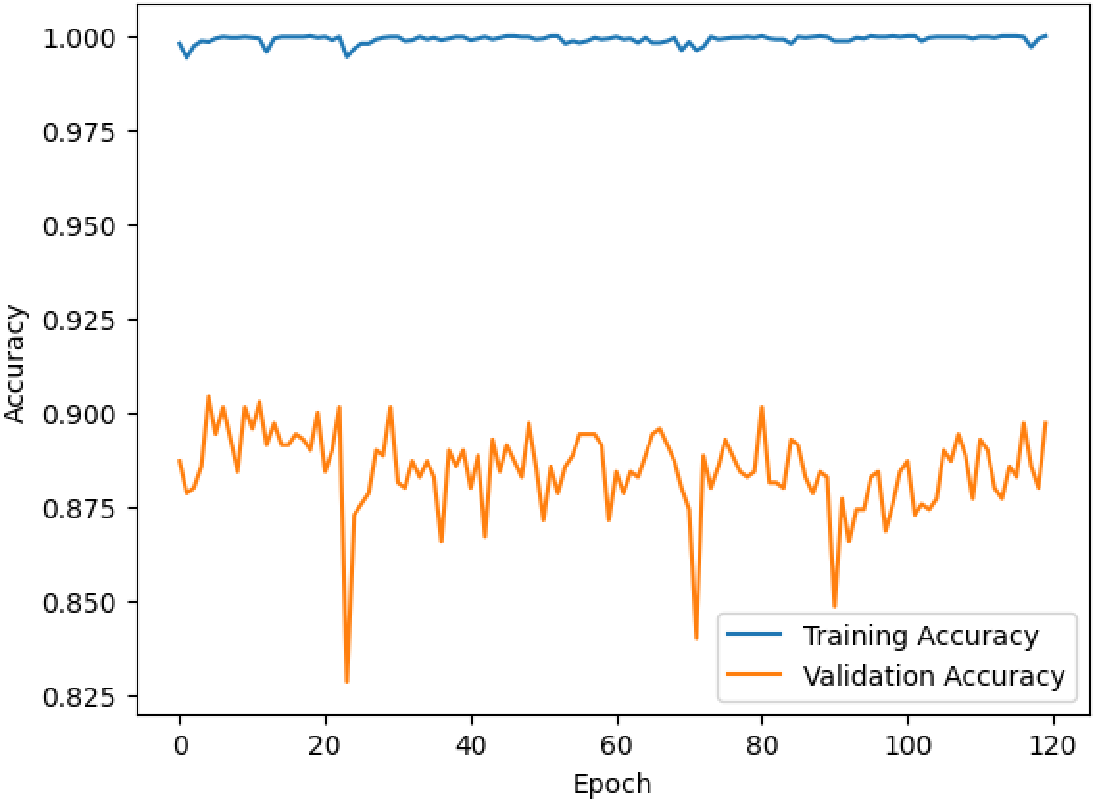
```
from keras.models import load_model
# Load the saved model
xception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/XFTSGDCP.hdf5')
```

[ ]:

```
from keras.optimizers import Adagrad
xception.compile(optimizer=Adagrad(learning_rate=0.01),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
AdaGrade_Histroy = xception.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[XFTADAGRADECP])
```

```
[ ]: plot_performance(AdaGrade_Histroy)
```





Optimizer ADADelta

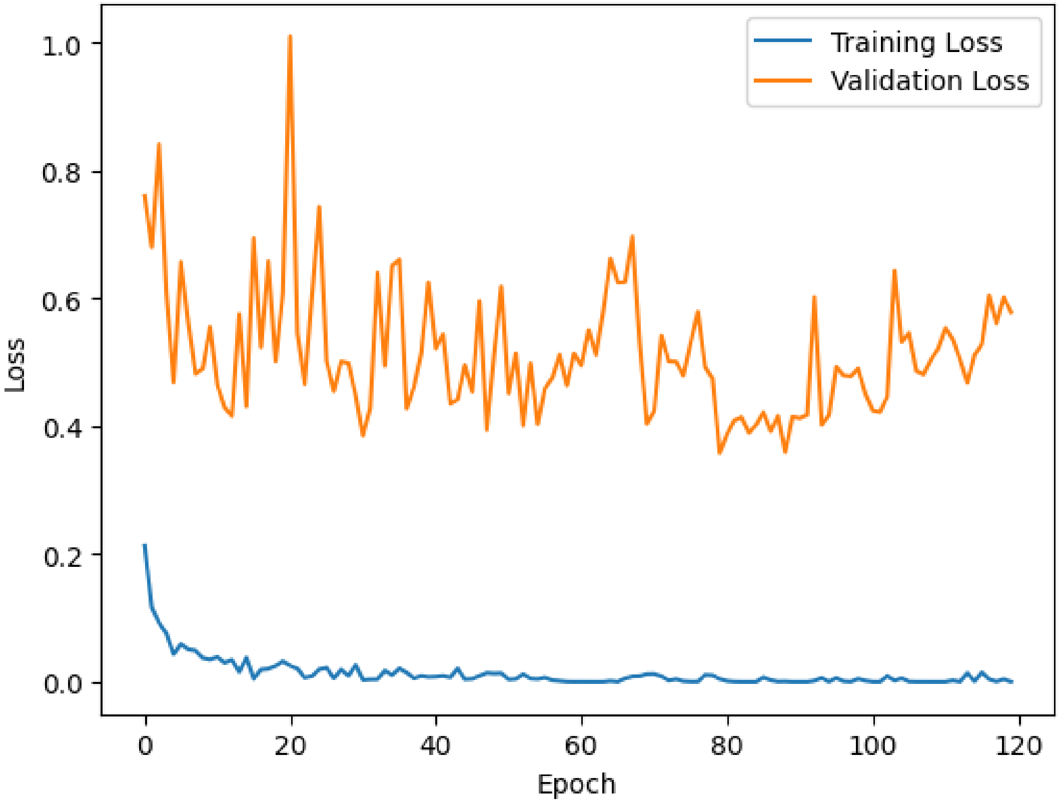
[ ]:
```
# define the model checkpoint callback
XFTADADELTACP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTADADELTACP.hdf5',
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

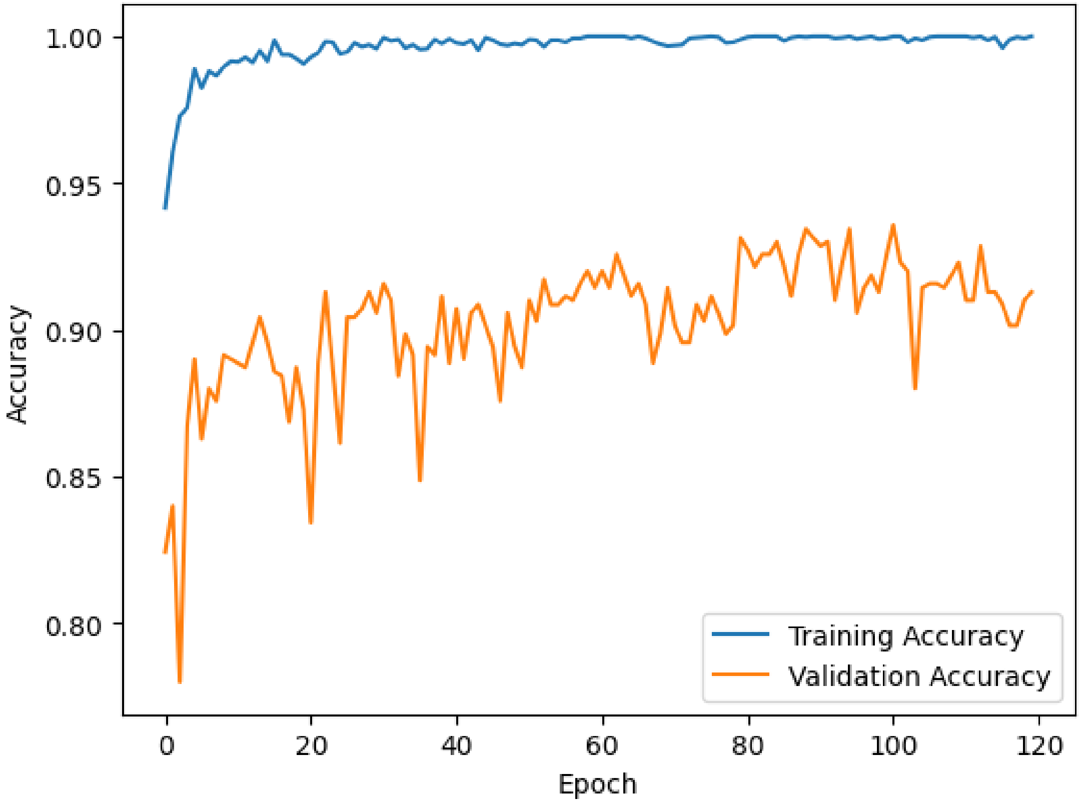
[ ]:
```
from keras.models import load_model
# Load the saved model
xception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/XFTADAGRADECP.hdf5')
```

[ ]:
```
from keras.optimizers import Adadelta
xception.compile(Adadelta(learning_rate=1.0), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
AdaDelta_History = xception.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[XFTADADELTACP])
```

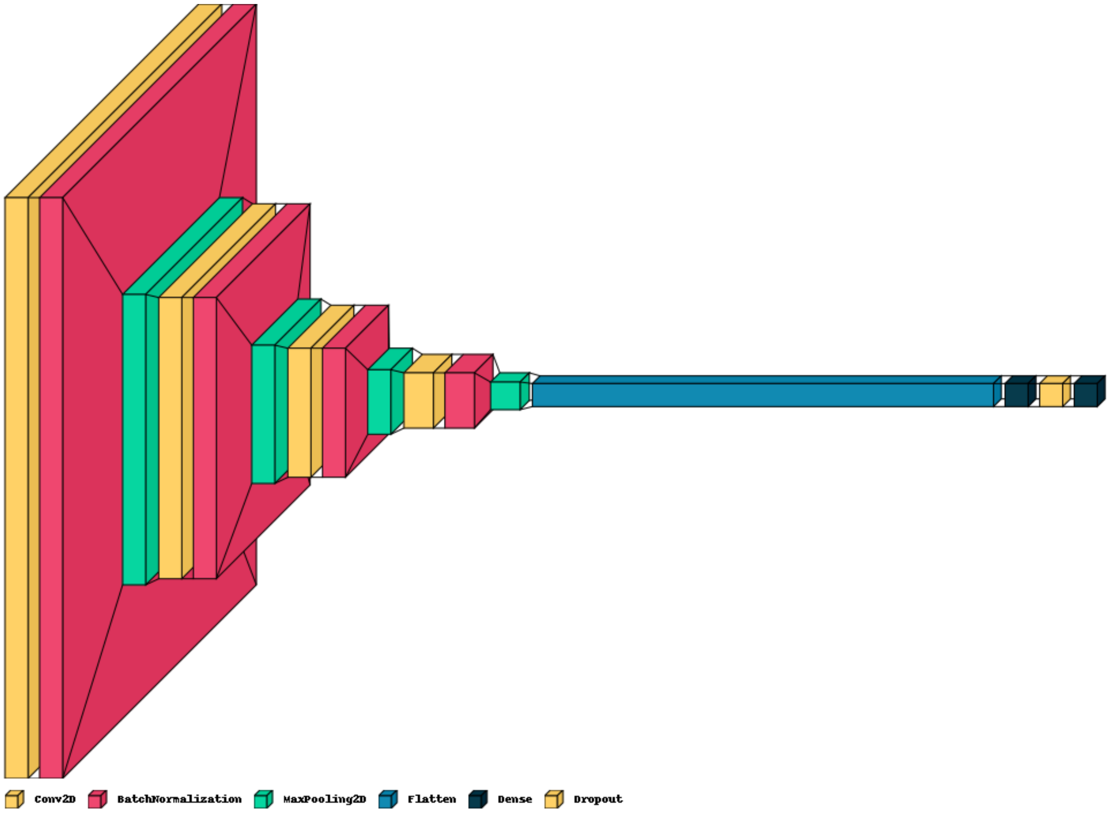
[ ]:

```
plot_performance(AdaDelta_History)
```





```
[ ]: import visualkeras
     from PIL import ImageFont
     visualkeras.layered_view(Model, legend=True)

[ ]:
```



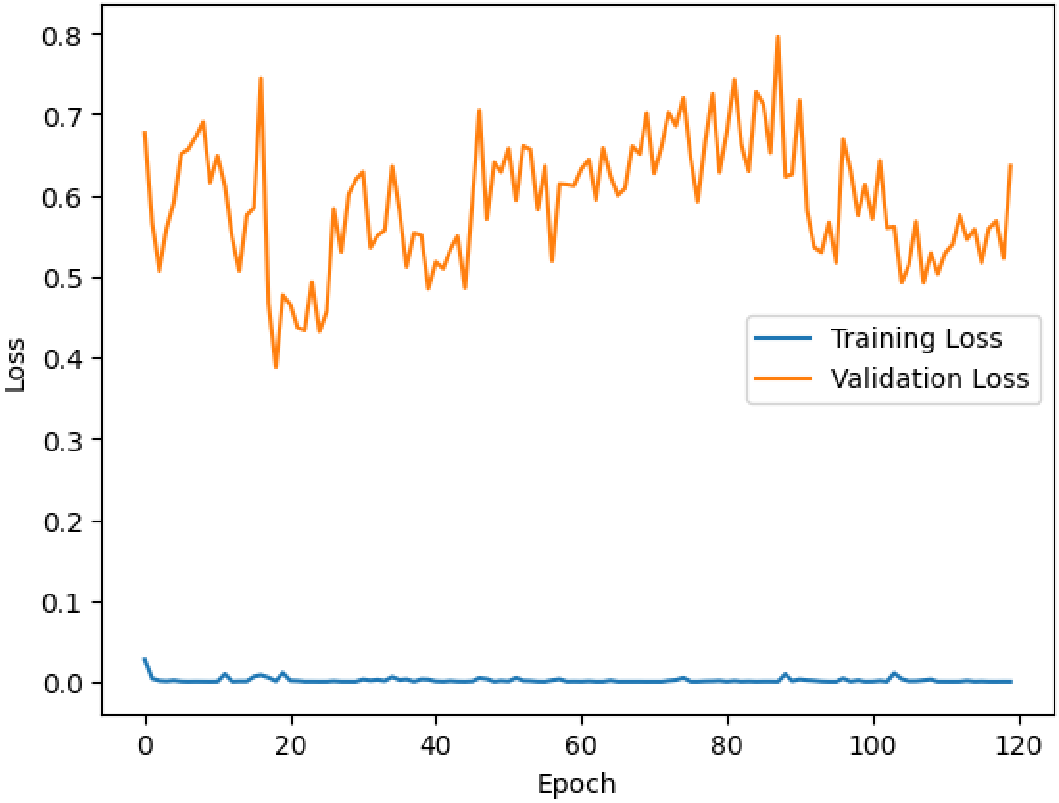
Optimizer ADAMax

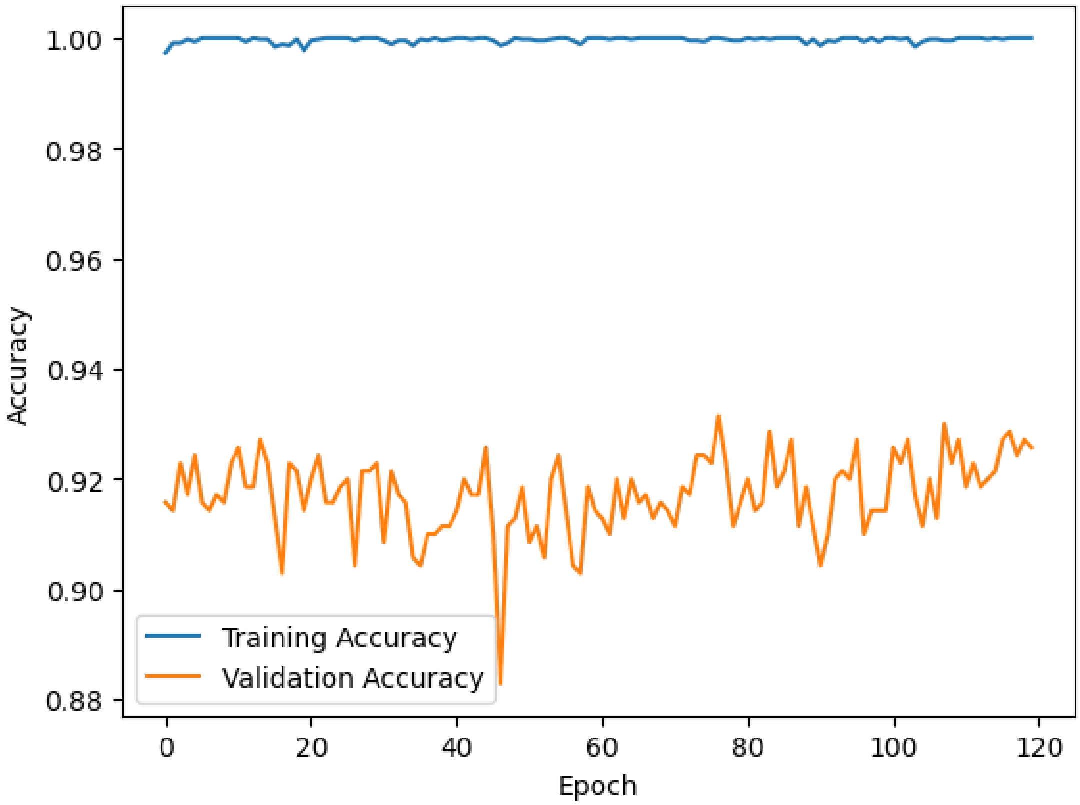
```
[ ]: # define the model checkpoint callback
     XFTADAMAXCP = ModelCheckpoint('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTADAMAXCP.hdf5',
                                   monitor='val_accuracy',
                                   verbose=1,
                                   save_best_only=True,
                                   mode='max')
```

```
[ ]: from keras.models import load_model
     # Load the saved model
     xception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/XFTADADELTACP.hdf5')
```

```
[ ]: from keras.optimizers import Adamax
     xception.compile(optimizer=Adamax(learning_rate=0.001),␣
      ↪loss='categorical_crossentropy', metrics=['accuracy'])
     Adamax_History = xception.fit(x=training_batches,␣
      ↪validation_data=validation_batches, epochs=120, verbose=2,␣
      ↪callbacks=[XFTADAMAXCP])
```

```
[ ]: plot_performance(Adamax_History)
```





Optimizer Nadam

[ ]:

```
# define the model checkpoint callback
XFTNADAMCP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTNADAMCP.hdf5',
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

[ ]:

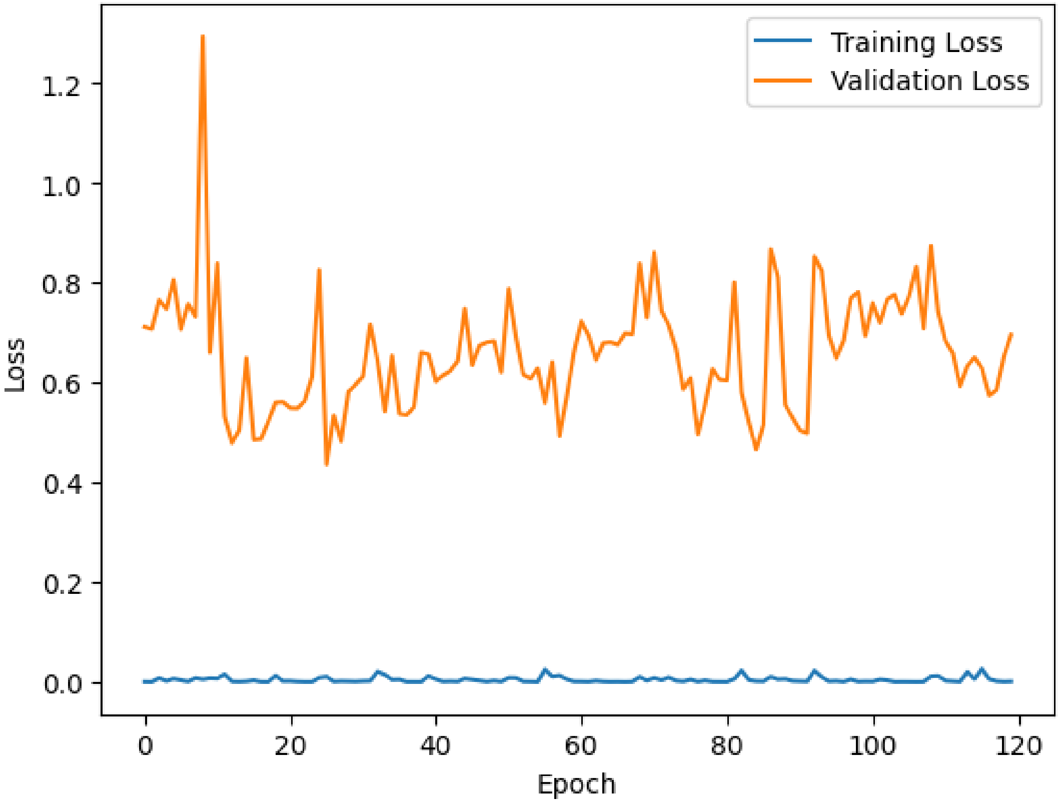
```
from keras.models import load_model
# Load the saved model
xception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/XFTADAMAXCP.hdf5')
```

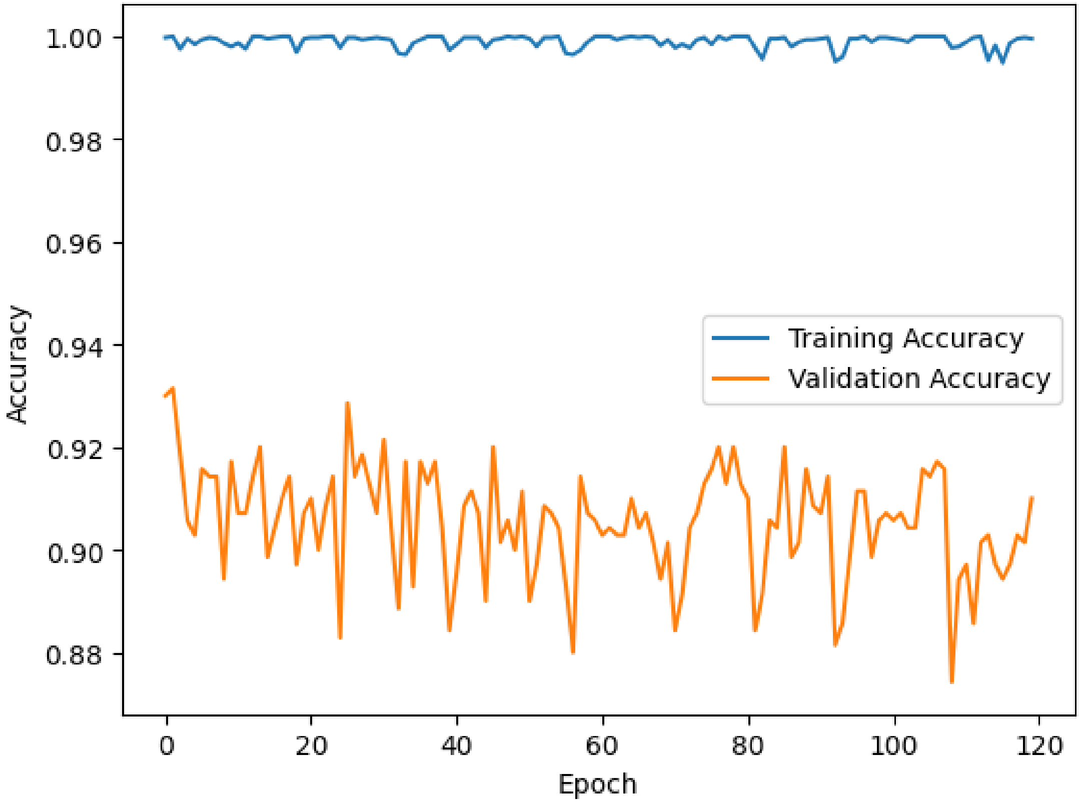
[ ]:

```
from keras.optimizers import Nadam
xception.compile(optimizer=Nadam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
Nadam_History = xception.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[XFTNADAMCP])
```

[ ]:

```
plot_performance(Nadam_History)
```





4.2.2 Fine-Tuning Inception V3

Optimizer SGD

```
[ ]: # define the model checkpoint callback
     IFTSGDCP = ModelCheckpoint('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTSGDCP.hdf5',
                                  monitor='val_accuracy',
                                  verbose=1,
                                  save_best_only=True,
                                  mode='max')
```

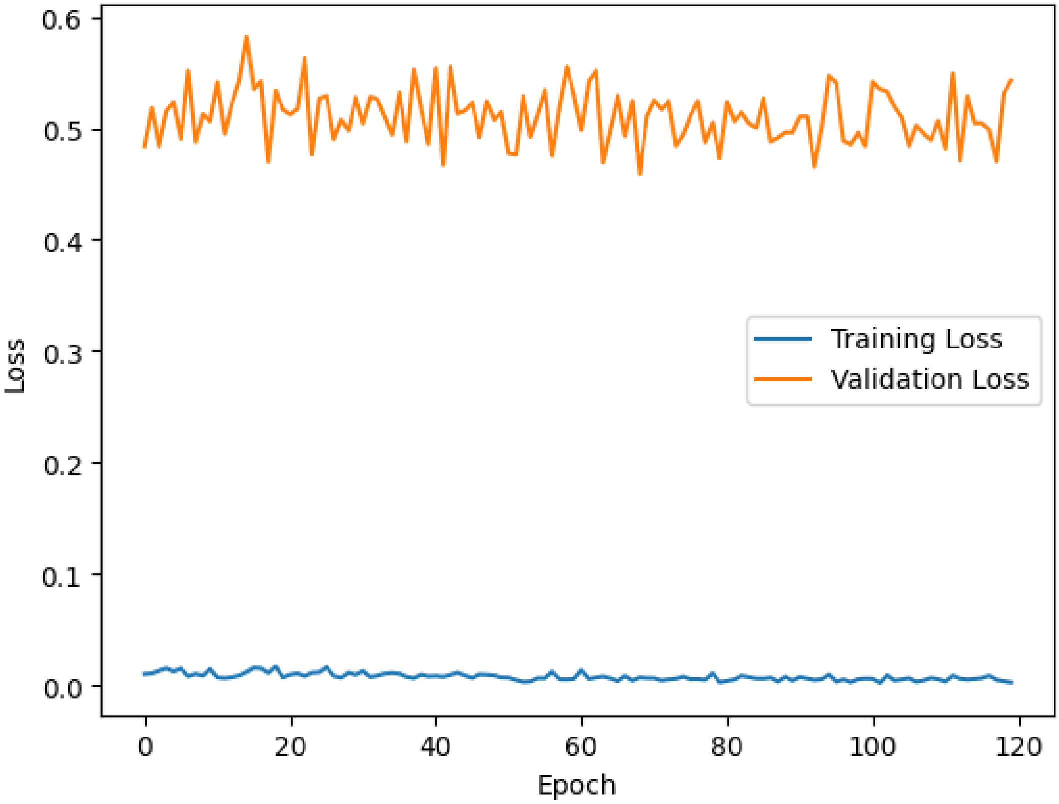
```
[ ]: #Load model
     '''Load default xception check point as the optimizer fine tnuing comes out to␣
      ↪be over fitting'''

     from keras.models import load_model

     # Load the saved model
     inception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/inception.hdf5')
```

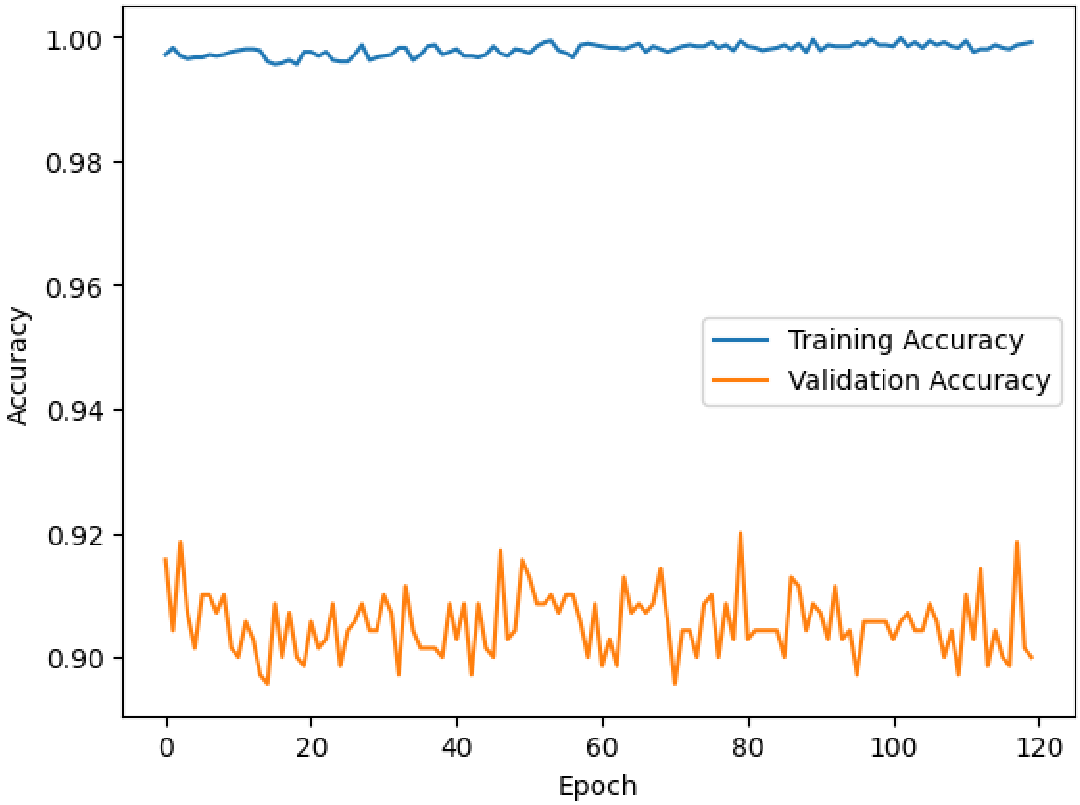
[ ]:

```
inception.compile(optimizer=SGD(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
SGD_Histroy_I = inception.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[IFTSGDCP])
```

[ ]:

```
plot_performance(SGD_Histroy_I)
```





Optimizer ADAGrade

[ ]:

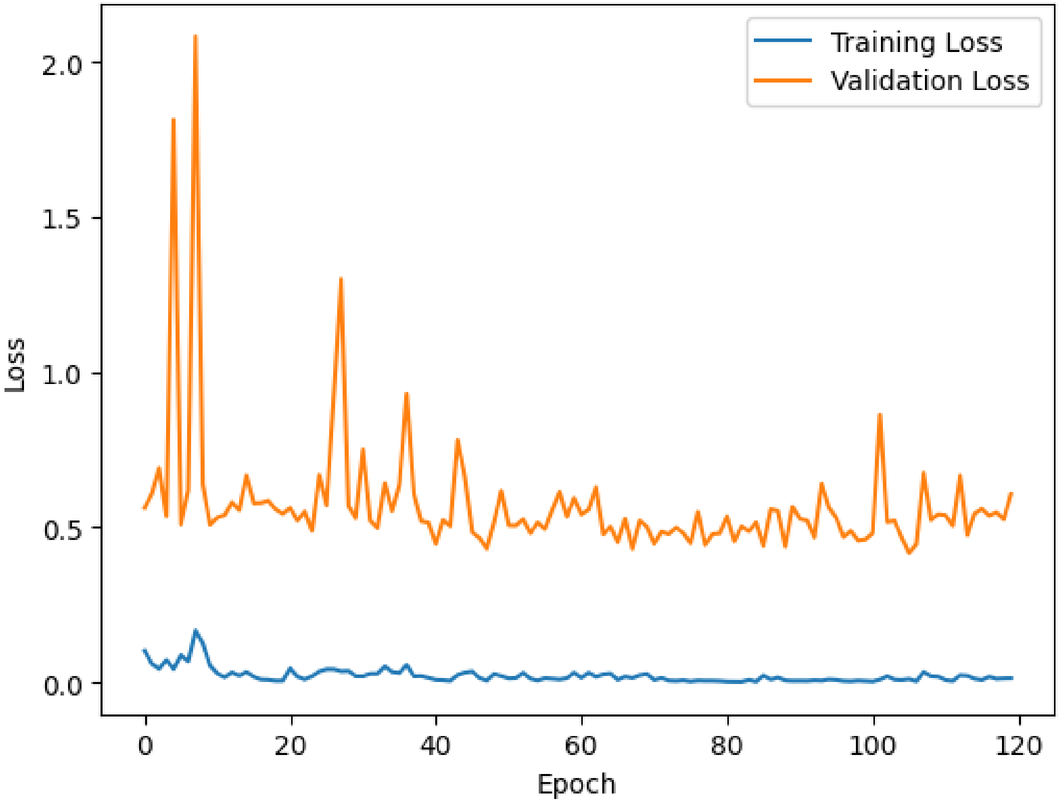
```
# define the model checkpoint callback
IFTADAGRADECP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTADAGRADECP.hdf5',
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

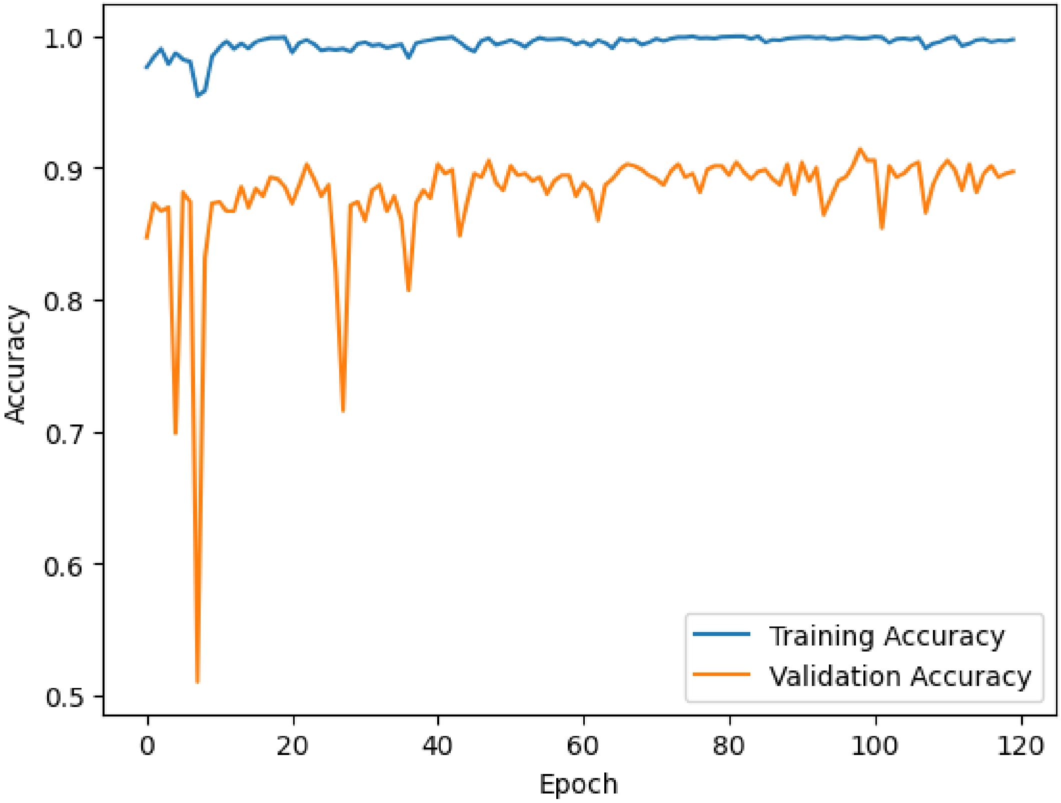
[ ]:

```
#Load model
'''Load default xception check point as the optimizer fine tnuing comes out to␣
 ↪be over fitting'''

from keras.models import load_model

# Load the saved model
inception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/IFTSGDCP.hdf5')
```

```
[ ]: from keras.optimizers import Adagrad
     inception.compile(optimizer=Adagrad(learning_rate=0.01),␣
      ↪loss='categorical_crossentropy', metrics=['accuracy'])
     AdaGrade_Histroy_I = inception.fit(x=training_batches,␣
      ↪validation_data=validation_batches, epochs=120, verbose=2,␣
      ↪callbacks=[IFTADAGRADECP])
```

```
[ ]: plot_performance(AdaGrade_Histroy_I)
```





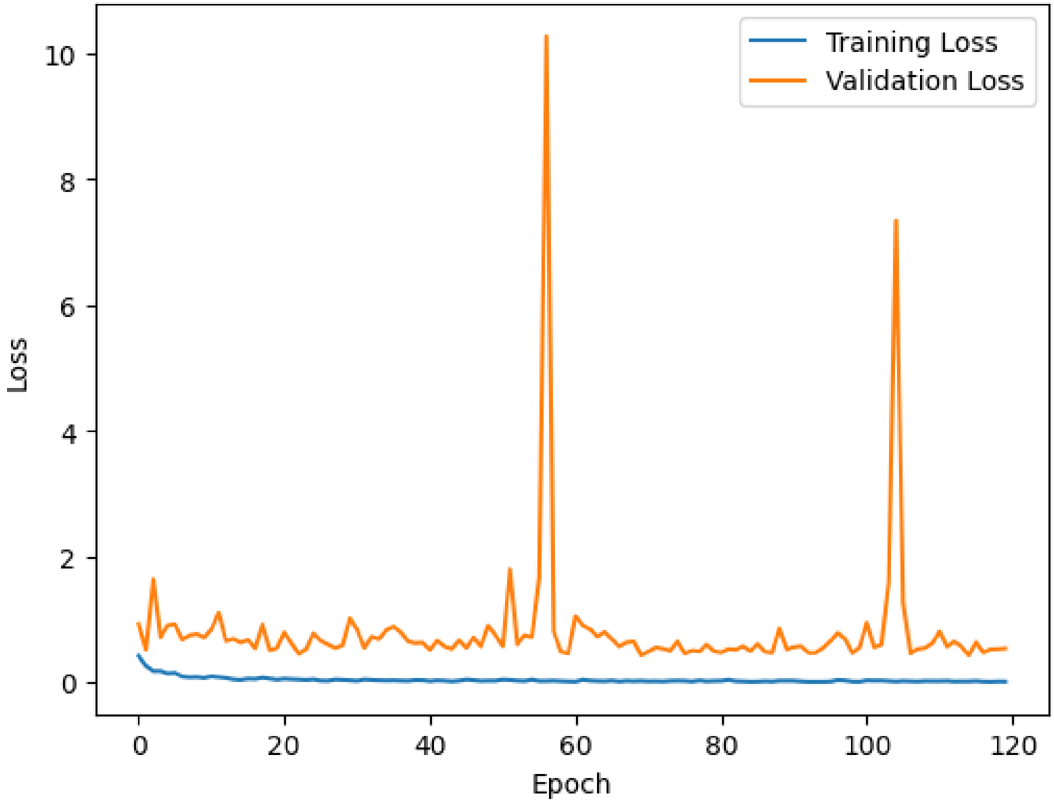
Optimizer ADADelta

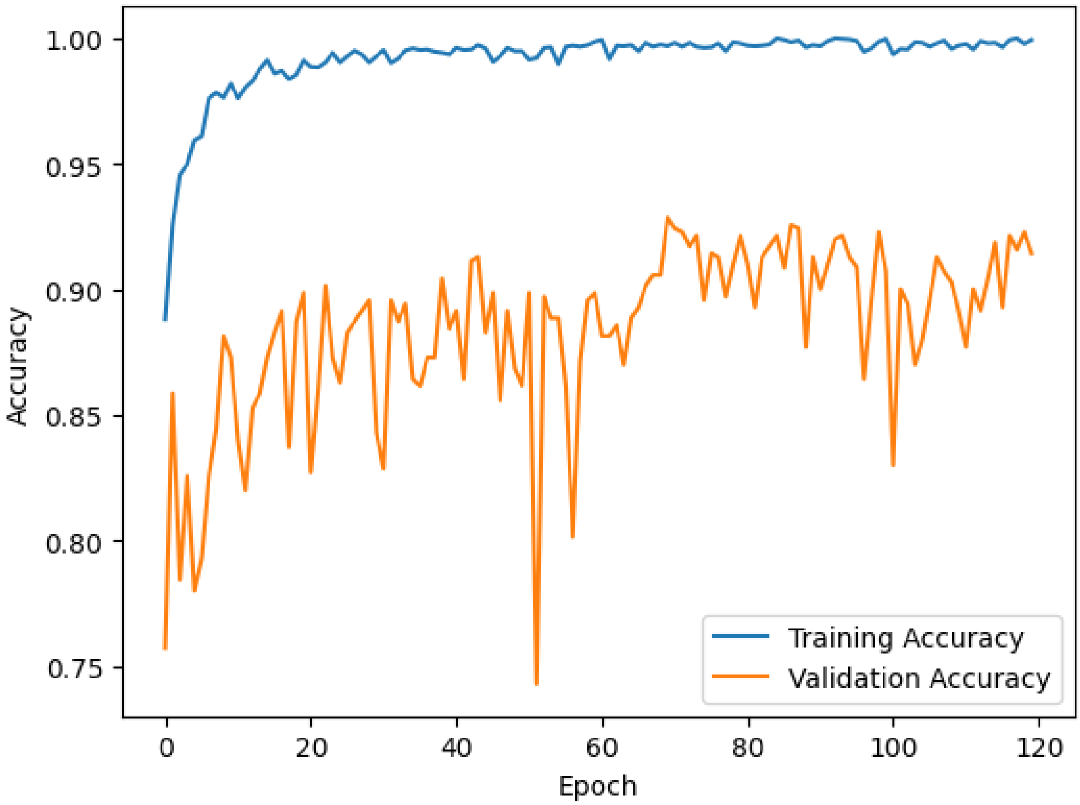
```
[ ]: # define the model checkpoint callback
IFTADADELTACP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTADADELTACP.hdf5',
                              monitor='val_accuracy',
                              verbose=1,
                              save_best_only=True,
                              mode='max')
```

```
[ ]: #Load model
'''Load default xception check point as the optimizer fine tnuing comes out to␣
 ↪be over fitting'''

from keras.models import load_model

# Load the saved model
inception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/IFTADAGRADECP.hdf5')
```

```
[ ]: from keras.optimizers import Adadelta
     inception.compile(Adadelta(learning_rate=1.0), loss='categorical_crossentropy',␣
      ↪metrics=['accuracy'])
     AdaDelta_History_I = inception.fit(x=training_batches,␣
      ↪validation_data=validation_batches, epochs=120, verbose=2,␣
      ↪callbacks=[IFTADADELTACP])
```

```
[ ]: plot_performance(AdaDelta_History_I)
```





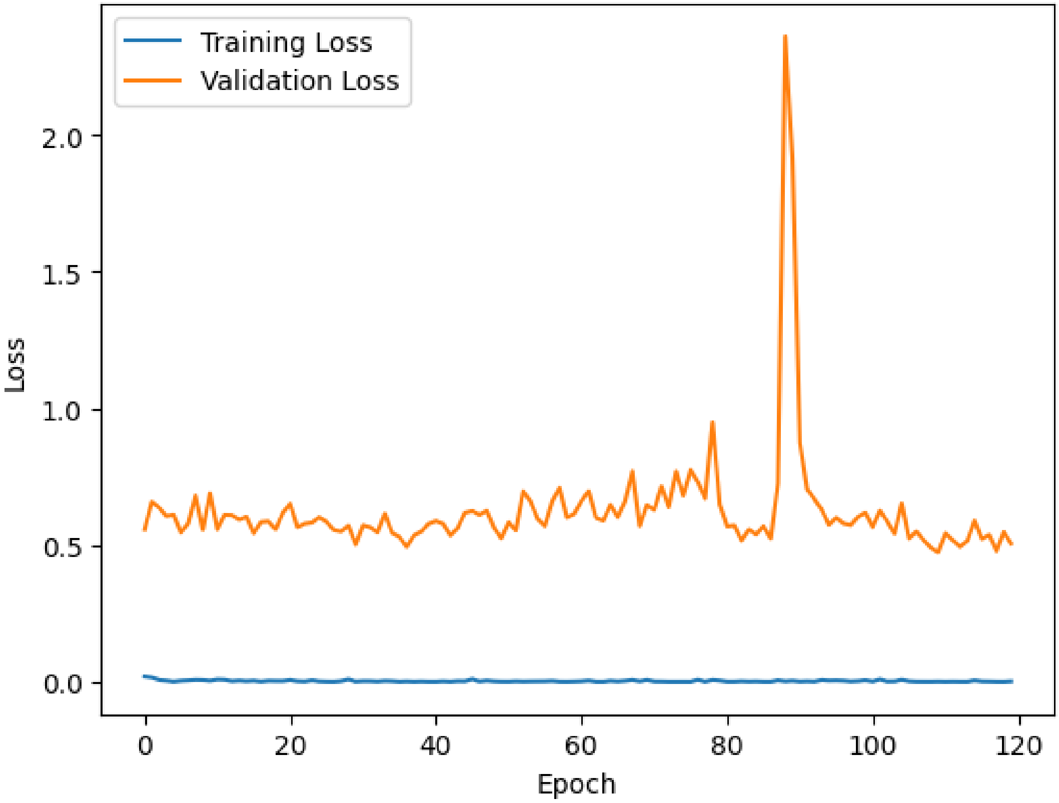
Optimizer ADAMax

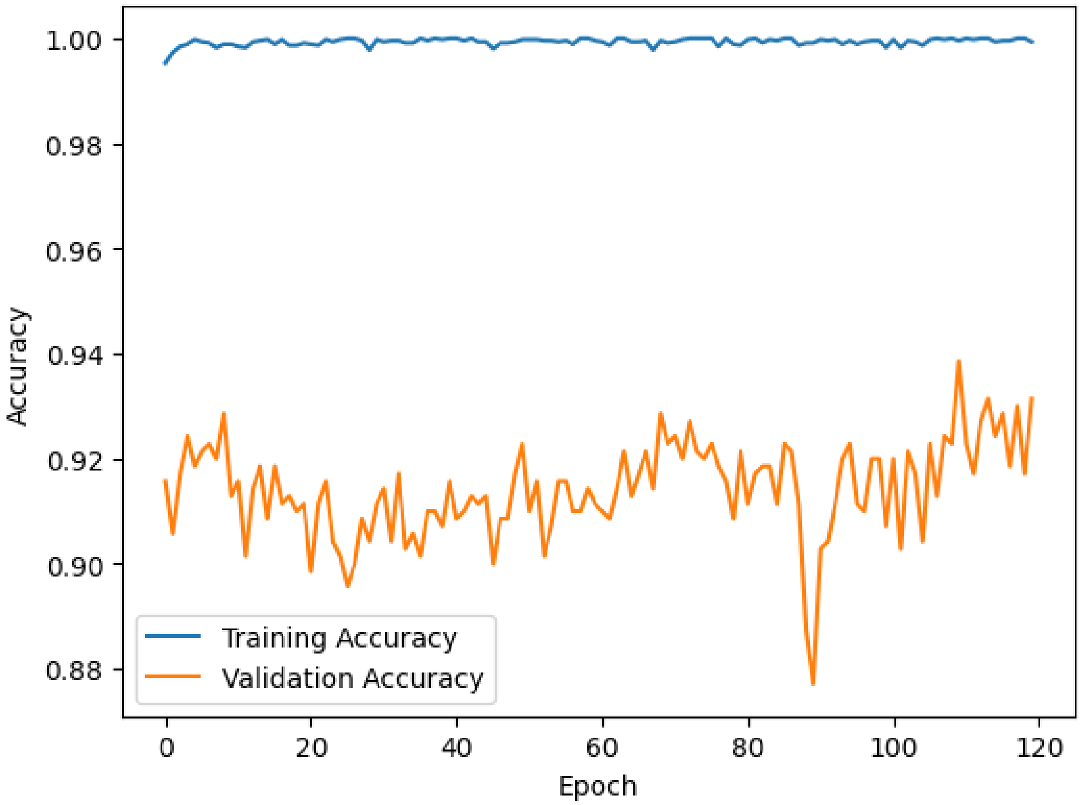
```
[ ]: # define the model checkpoint callback
     IFTADAMAXCP = ModelCheckpoint('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTADAMAXCP.hdf5',
                                  monitor='val_accuracy',
                                  verbose=1,
                                  save_best_only=True,
                                  mode='max')
```

```
[ ]: #Load model
     '''Load default xception check point as the optimizer fine tnuing comes out to␣
      ↪be over fitting'''

     from keras.models import load_model

     # Load the saved model
     inception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/IFTADADELTACP.hdf5')
```

[ ]:
```
from keras.optimizers import Adamax
inception.compile(optimizer=Adamax(learning_rate=0.001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
Adamax_History_I = inception.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[IFTADAMAXCP])
```

[ ]:
```
plot_performance(Adamax_History_I)
```





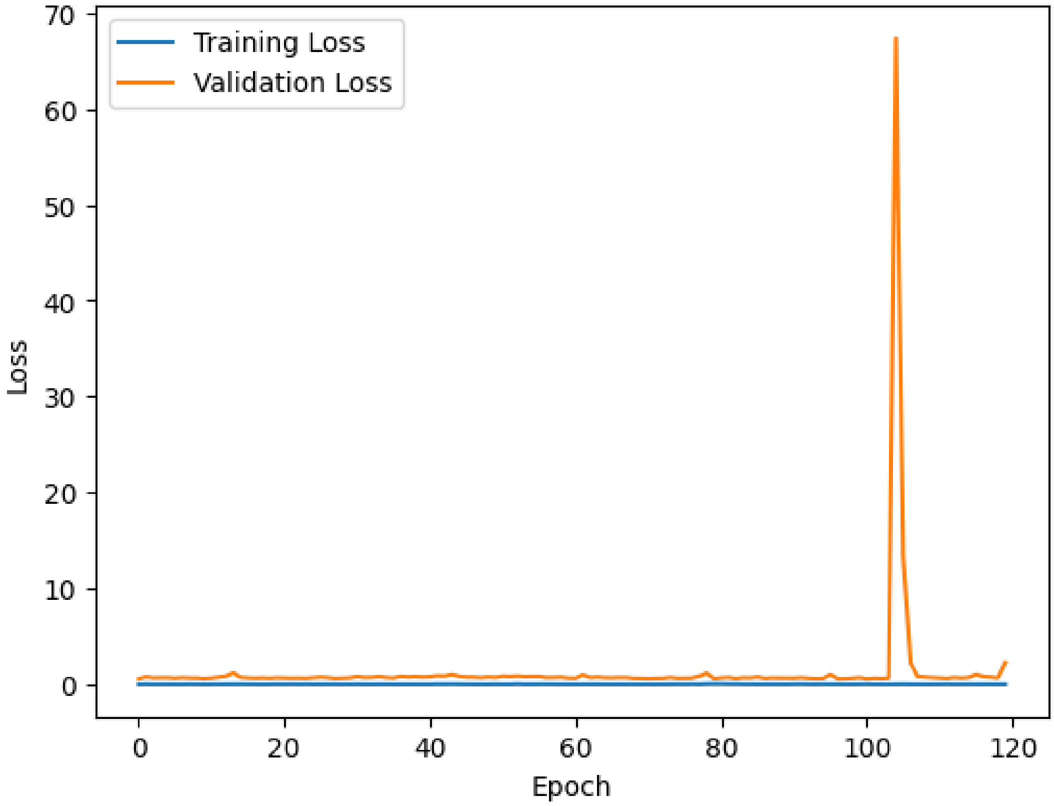
Optimizer Nadam

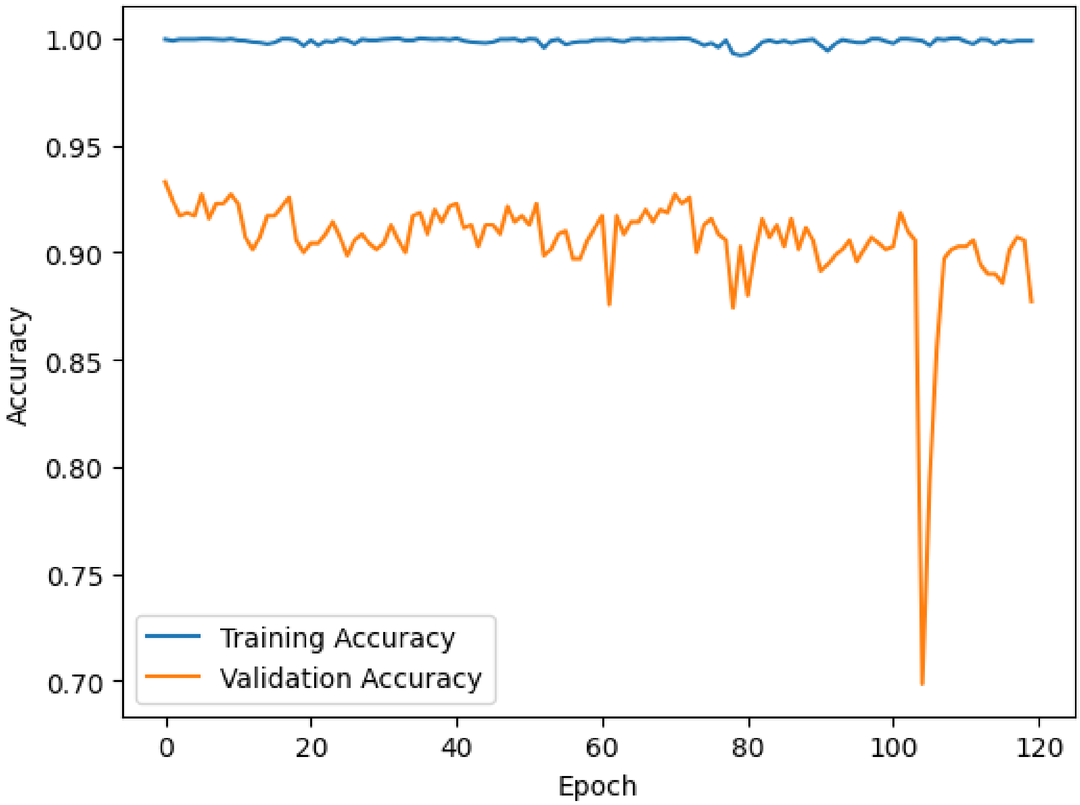
```
[ ]: # define the model checkpoint callback
     IFTNADAMCP = ModelCheckpoint('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTNADAMCP.hdf5',
                                  monitor='val_accuracy',
                                  verbose=1,
                                  save_best_only=True,
                                  mode='max')
```

```
[ ]: #Load model
     '''Load default xception check point as the optimizer fine tnuing comes out to␣
      ↪be over fitting'''

     from keras.models import load_model

     # Load the saved model
     inception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/IFTADAMAXCP.hdf5')
```

```
[ ]: from keras.optimizers import Nadam
     inception.compile(optimizer=Nadam(learning_rate=0.0001),␣
      ↪loss='categorical_crossentropy', metrics=['accuracy'])
     Nadam_History_I = inception.fit(x=training_batches,␣
      ↪validation_data=validation_batches, epochs=120, verbose=2,␣
      ↪callbacks=[IFTNADAMCP])
```

```
[ ]: plot_performance(Nadam_History_I)
```





4.2.3 Fine-Tuning Sequential CNN

Optimizer SGD

```
[ ]: # define the model checkpoint callback
     MFTSGDCP = ModelCheckpoint('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTSGDCP.hdf5',
                                  monitor='val_accuracy',
                                  verbose=1,
                                  save_best_only=True,
                                  mode='max')
```

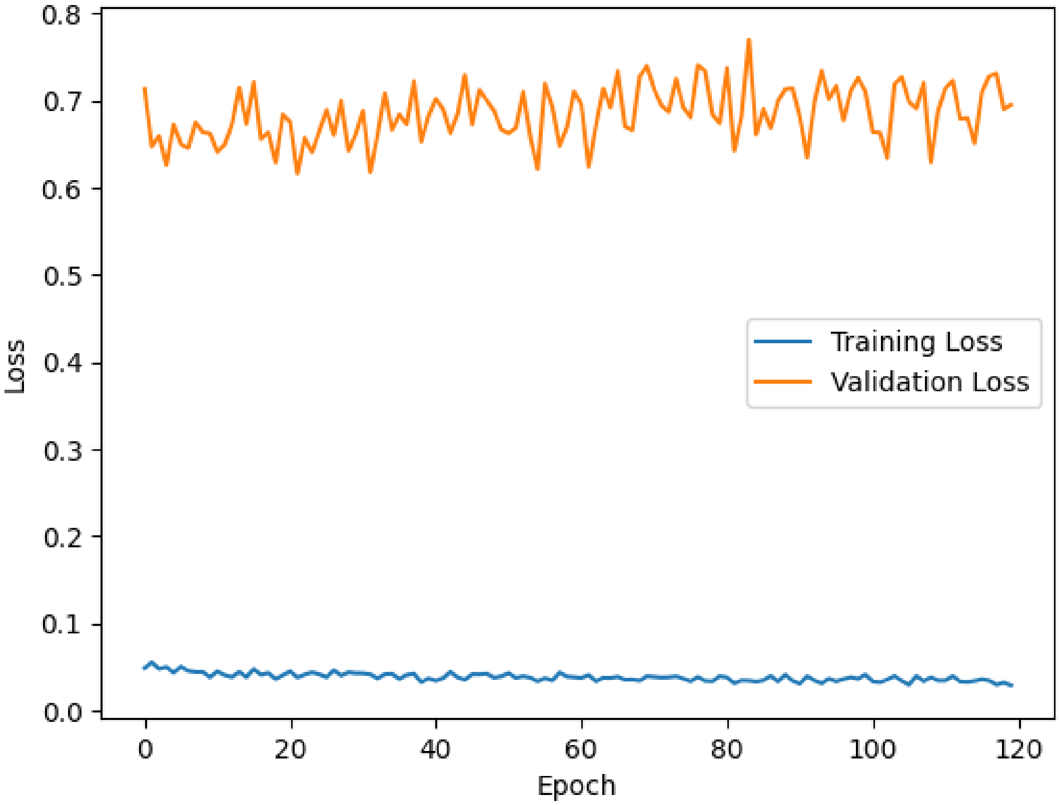
```
[ ]: #Load model
     '''Load default xception check point as the optimizer fine tnuing comes out to␣
      ↪be over fitting'''

     from keras.models import load_model

     # Load the saved model
     model = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/custom_sequence.hdf5')
```

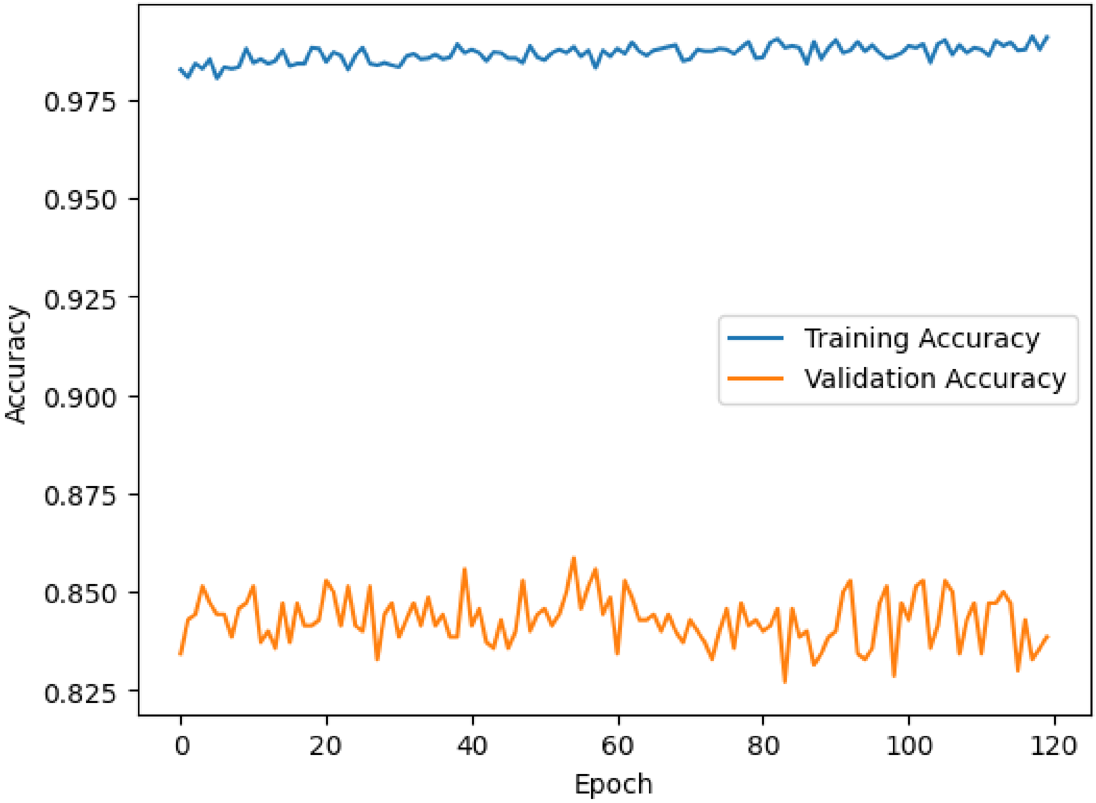
[ ]:

```
model.compile(optimizer=SGD(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
SGD_Histroy_I = model.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[MFTSGDCP])
```

[ ]:

```
plot_performance(SGD_Histroy_I)
```





Optimizer ADAGrade

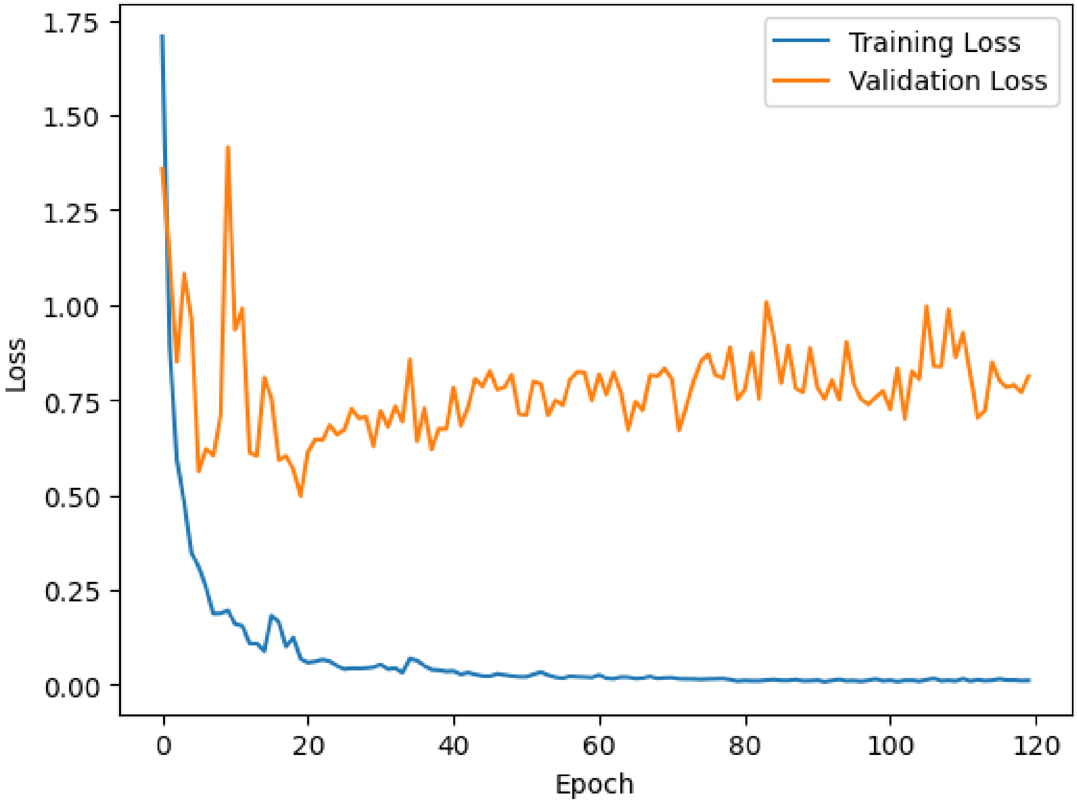
```
[ ]: # define the model checkpoint callback
MFTADAGRADECP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTADAGRADECP.hdf5',
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

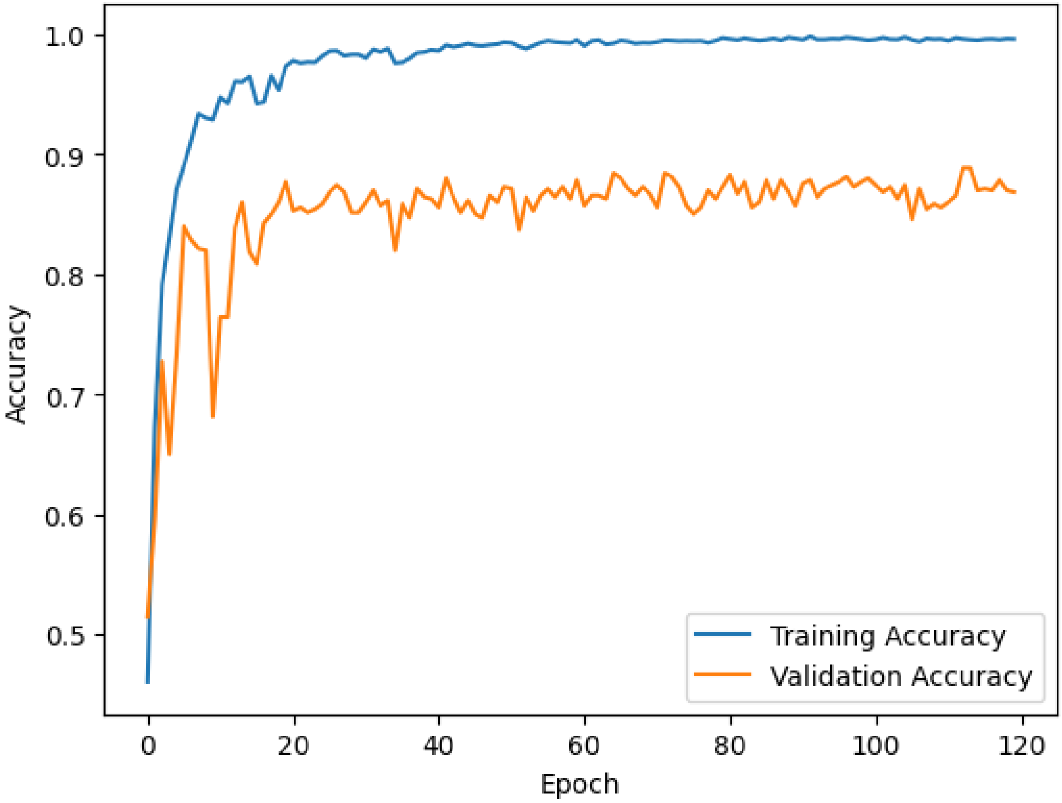
```
[ ]: # Load the saved model
model = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/MFTSGDCP.hdf5')
```

```
[ ]: from keras.optimizers import Adagrad
model.compile(optimizer=Adagrad(learning_rate=0.01),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
AdaGrade_Histroy_M = model.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[MFTADAGRADECP])
```

[ ]:

```
plot_performance(AdaGrade_Histroy_M)
```





Optimizer ADADelta

[ ]:

```
# define the model checkpoint callback
MFTADADELTACP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTADADELTACP.hdf5',
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

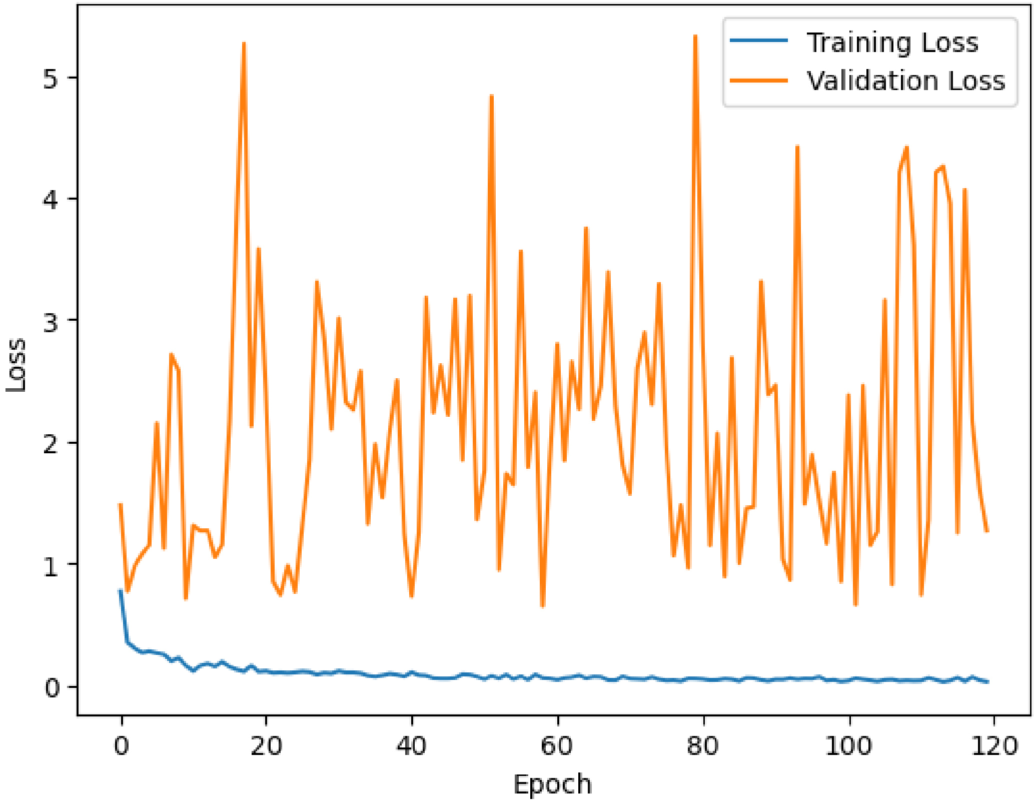
[ ]:

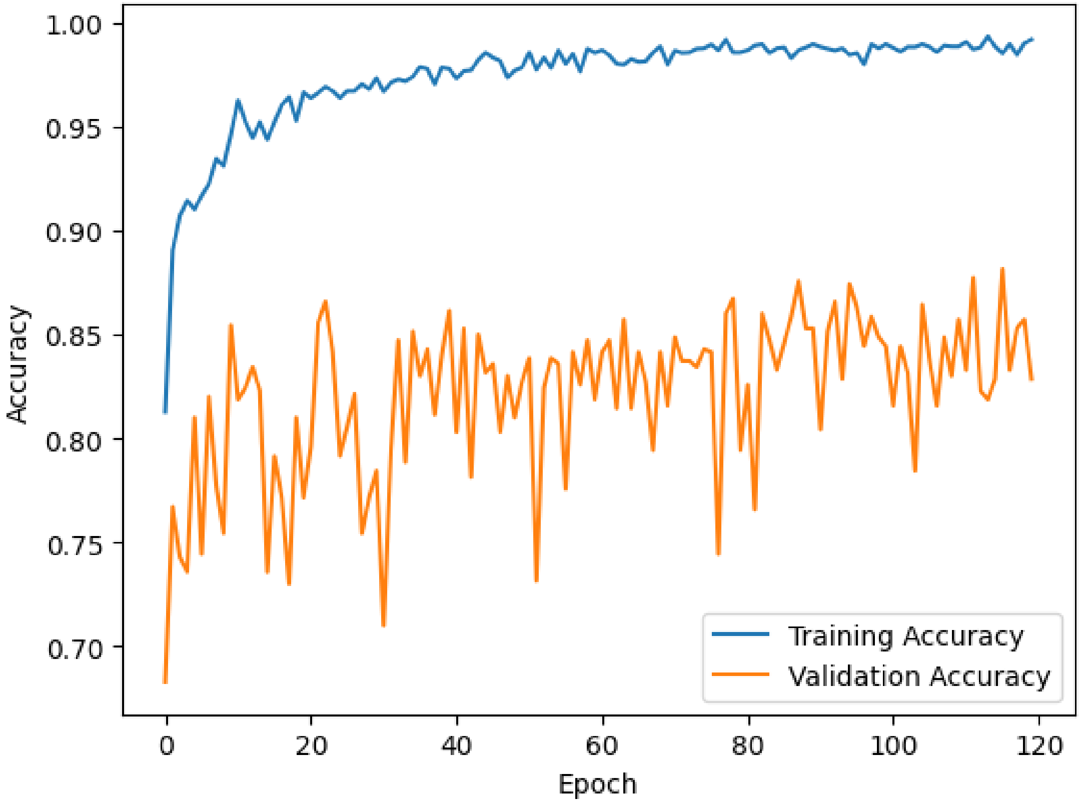
```
# Load the saved model
model = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/MFTADAGRADECP.hdf5')
```

[ ]:

```
from keras.optimizers import Adadelta
model.compile(Adadelta(learning_rate=1.0), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
AdaDelta_History_M = model.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[MFTADADELTACP])
```

```
[ ]: plot_performance(AdaDelta_History_M)
```





Optimizer ADAMax

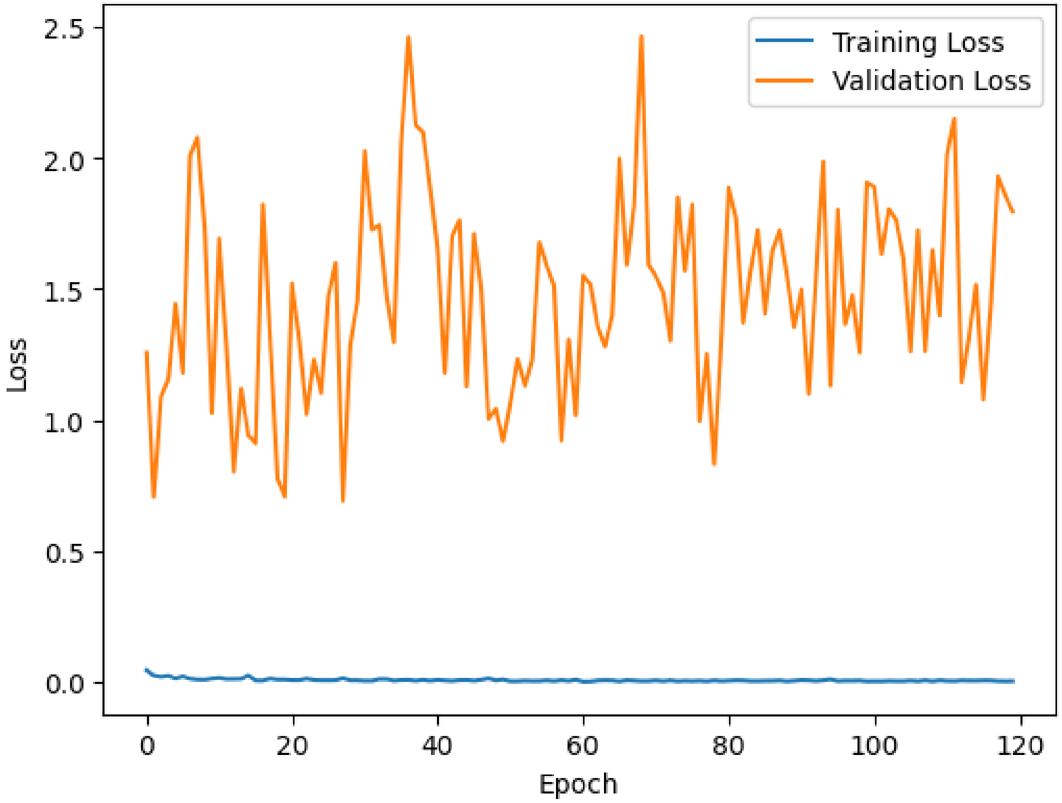
[ ]:
```
# define the model checkpoint callback
MFTADAMAXCP = ModelCheckpoint('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTADAMAXCP.hdf5',
                              monitor='val_accuracy',
                              verbose=1,
                              save_best_only=True,
                              mode='max')
```

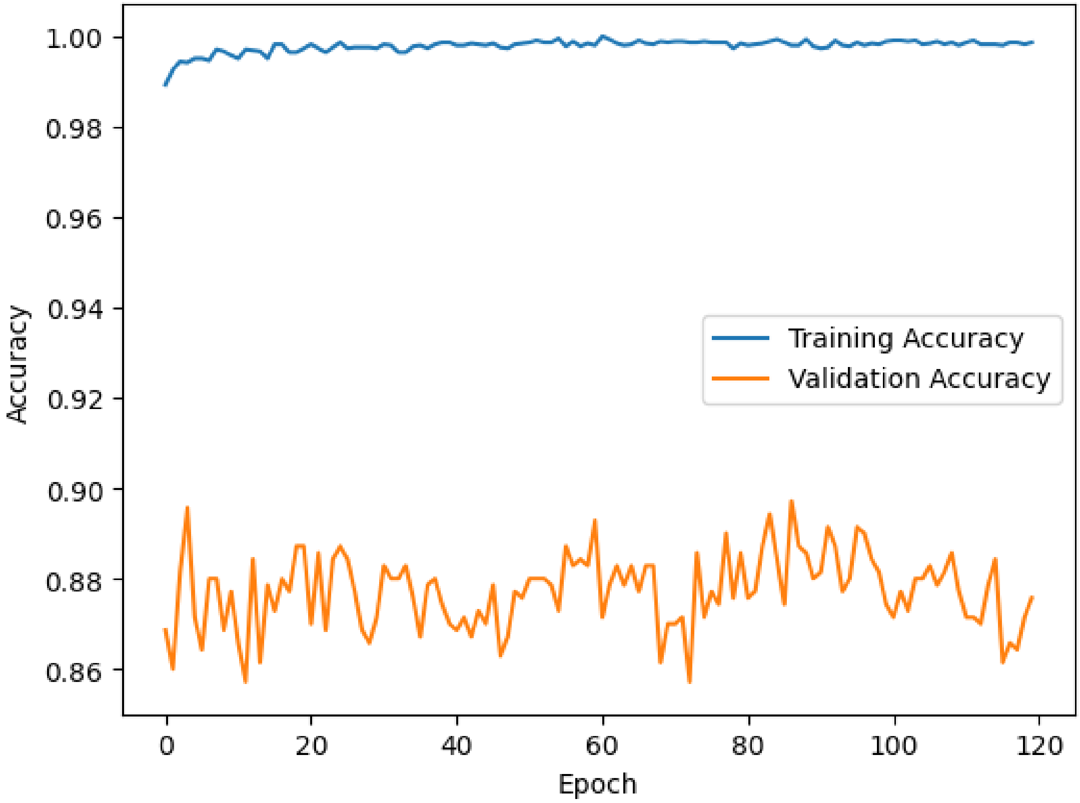
[ ]:
```
# Load the saved model
model = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/MFTADADELTACP.hdf5')
```

[ ]:
```
from keras.optimizers import Adamax
model.compile(optimizer=Adamax(learning_rate=0.001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
Adamax_History_I = model.fit(x=training_batches,␣
 ↪validation_data=validation_batches, epochs=120, verbose=2,␣
 ↪callbacks=[MFTADAMAXCP])
```

[ ]:

```
plot_performance(Adamax_History_I)
```





Optimizer Nadam

```
[ ]: # define the model checkpoint callback
     MFTNADAMCP = ModelCheckpoint('/content/drive/MyDrive/
       ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTNADAMCP.hdf5',
                                  monitor='val_accuracy',
                                  verbose=1,
                                  save_best_only=True,
                                  mode='max')
```

```
[ ]: #Load model
     '''Load default xception check point as the optimizer fine tnuing comes out to␣
       ↪be over fitting'''

     from keras.models import load_model

     # Load the saved model
     model = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
       ↪Models/MFTADAMAXCP.hdf5')
```

```
[ ]: from keras.optimizers import Nadam
     model.compile(optimizer=Nadam(learning_rate=0.0001),␣
      ↪loss='categorical_crossentropy', metrics=['accuracy'])
     Nadam_History_M = model.fit(x=training_batches,␣
      ↪validation_data=validation_batches, epochs=120, verbose=2,␣
      ↪callbacks=[MFTNADAMCP])
```

```
[ ]: plot_performance(Nadam_History_M)
```

4.3 3.3 Testing Models

```
[ ]: test_imgs, test_labels = next(testing_batches)
     plotImages(test_imgs)
```

```
[ ]: testing_batches.class_indices
```

```
[ ]: {'anger': 0,
      'disgust': 1,
      'fear': 2,
      'happiness': 3,
      'neutral': 4,
      'sadness': 5,
      'surprise': 6}
```

4.3.1 Xception Test

```
[ ]: # Load the saved model
     Xception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/xception.hdf5')
     XFTADADELTACP = load_model('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTADADELTACP.hdf5')
     XFTADAGRADECP = load_model('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTADAGRADECP.hdf5')
     XFTNADAMCP = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/XFTNADAMCP.hdf5')
     XFTRMSPROPCP = load_model('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTRMSPROPCP.hdf5')
     XFTADAMAXCP = load_model('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/XFTADAMAXCP.hdf5')
```

```
XFTSGDCP = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/XFTSGDCP.hdf5')

Xception_Versions = [Xception, XFTADADELTACP, XFTADAGRADECP, XFTNADAMCP,␣
 ↪XFTRMSPROPCP, XFTADAMAXCP, XFTSGDCP]
```

[ ]:
```
for i, model in enumerate(Xception_Versions):
    print(f'Model no: {i}')
    test_loss, test_acc = model.evaluate(validation_batches, verbose=1)
    print(f'Validation accuracy:', test_acc)
    test_loss, test_acc = model.evaluate(testing_batches, verbose=1)
    print(f'Testing accuracy:', test_acc)
    print('-'*80+'\n')
```

```
Model no: 0
11/11 [==============================] - 229s 22s/step - loss: 0.5082 -
accuracy: 0.9057
Validation accuracy: 0.9057142734527588
11/11 [==============================] - 223s 22s/step - loss: 0.5468 -
accuracy: 0.8986
Testing accuracy: 0.8985714316368103
--------------------------------------------------------------------------------

Model no: 1
11/11 [==============================] - 51s 5s/step - loss: 0.4660 - accuracy:
0.9243
Validation accuracy: 0.9242857098579407
11/11 [==============================] - 48s 4s/step - loss: 0.5645 - accuracy:
0.9071
Testing accuracy: 0.9071428775787354
--------------------------------------------------------------------------------

Model no: 2
11/11 [==============================] - 49s 4s/step - loss: 0.5326 - accuracy:
0.8929
Validation accuracy: 0.8928571343421936
11/11 [==============================] - 47s 4s/step - loss: 0.5542 - accuracy:
0.8843
Testing accuracy: 0.8842856884002686
--------------------------------------------------------------------------------

Model no: 3
11/11 [==============================] - 49s 4s/step - loss: 0.7999 - accuracy:
0.9229
Validation accuracy: 0.9228571653366089
11/11 [==============================] - 47s 4s/step - loss: 0.8077 - accuracy:
0.9071
```

```
Testing accuracy: 0.9071428775787354
--------------------------------------------------------------------------------

Model no: 4
11/11 [==============================] - 49s 4s/step - loss: 2.7988 - accuracy:
0.9014
Validation accuracy: 0.9014285802841187
11/11 [==============================] - 47s 4s/step - loss: 3.1046 - accuracy:
0.8814
Testing accuracy: 0.881428599357605
--------------------------------------------------------------------------------

Model no: 5
11/11 [==============================] - 55s 5s/step - loss: 0.6461 - accuracy:
0.9271
Validation accuracy: 0.927142858505249
11/11 [==============================] - 47s 4s/step - loss: 0.7111 - accuracy:
0.9157
Testing accuracy: 0.9157142639160156
--------------------------------------------------------------------------------

Model no: 6
11/11 [==============================] - 48s 4s/step - loss: 0.4638 - accuracy:
0.9043
Validation accuracy: 0.904285728931427
11/11 [==============================] - 47s 4s/step - loss: 0.5835 - accuracy:
0.9000
Testing accuracy: 0.8999999761581421
--------------------------------------------------------------------------------
```

4.3.2 Inception Test

[ ]:

```
# Load the saved model
Inception = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/inception.hdf5')
IFTADADELTACP = load_model('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTADADELTACP.hdf5')
IFTADAGRADECP = load_model('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTADAGRADECP.hdf5')
IFTADAMAXCP = load_model('/content/drive/MyDrive/
 ↪Muhammad_Ali_Thesis_3059828_FER/Models/IFTADAMAXCP.hdf5')
IFTNADAMCP = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/IFTNADAMCP.hdf5')
IFTSGDCP = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
 ↪Models/IFTSGDCP.hdf5')
```

```
Inception_Versions = [Inception, IFTADADELTACP, IFTADAGRADECP, IFTNADAMCP,␣
 ↪IFTADAMAXCP, IFTSGDCP]
```

```
[ ]: for i, model in enumerate(Inception_Versions):
         print(f'Model no: {i}')
         test_loss, test_acc = model.evaluate(validation_batches, verbose=1)
         print(f'Validation accuracy:', test_acc)
         test_loss, test_acc = model.evaluate(testing_batches, verbose=1)
         print(f'Testing accuracy:', test_acc)
         print('-'*80+'\n')
```

```
Model no: 0
11/11 [==============================] - 1s 97ms/step - loss: 0.5913 - accuracy:
0.9000
Validation accuracy: 0.8999999761581421
11/11 [==============================] - 1s 102ms/step - loss: 0.6891 -
accuracy: 0.8800
Testing accuracy: 0.8799999952316284
--------------------------------------------------------------------------------

Model no: 1
11/11 [==============================] - 1s 110ms/step - loss: 0.4249 -
accuracy: 0.9286
Validation accuracy: 0.9285714030265808
11/11 [==============================] - 1s 101ms/step - loss: 0.7174 -
accuracy: 0.8871
Testing accuracy: 0.8871428370475769
--------------------------------------------------------------------------------

Model no: 2
11/11 [==============================] - 1s 98ms/step - loss: 0.5134 - accuracy:
0.9014
Validation accuracy: 0.9014285802841187
11/11 [==============================] - 1s 98ms/step - loss: 0.5938 - accuracy:
0.8914
Testing accuracy: 0.8914285898208618
--------------------------------------------------------------------------------

Model no: 3
11/11 [==============================] - 1s 119ms/step - loss: 0.5398 -
accuracy: 0.9343
Validation accuracy: 0.9342857003211975
11/11 [==============================] - 1s 99ms/step - loss: 0.8924 - accuracy:
0.9043
Testing accuracy: 0.904285728931427
--------------------------------------------------------------------------------
```

```
Model no: 4
11/11 [==============================] - 1s 98ms/step - loss: 0.5496 - accuracy:
0.9214
Validation accuracy: 0.9214285612106323
11/11 [==============================] - 1s 98ms/step - loss: 0.8466 - accuracy:
0.8914
Testing accuracy: 0.8914285898208618
--------------------------------------------------------------------------------

Model no: 5
11/11 [==============================] - 3s 261ms/step - loss: 0.5185 -
accuracy: 0.9129
Validation accuracy: 0.9128571152687073
11/11 [==============================] - 1s 109ms/step - loss: 0.5890 -
accuracy: 0.8914
Testing accuracy: 0.8914285898208618
--------------------------------------------------------------------------------
```

4.3.3 Sequential CNN Model Test

```
[ ]: # Load the saved model
     Model = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/model.hdf5')
     MFTADADELTACP = load_model('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTADADELTACP.hdf5')
     MFTADAGRADECP = load_model('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTADAGRADECP.hdf5')
     MFTADAMAXCP = load_model('/content/drive/MyDrive/
      ↪Muhammad_Ali_Thesis_3059828_FER/Models/MFTADAMAXCP.hdf5')
     MFTNADAMCP = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/MFTNADAMCP.hdf5')
     MFTSGDCP = load_model('/content/drive/MyDrive/Muhammad_Ali_Thesis_3059828_FER/
      ↪Models/MFTSGDCP.hdf5')

     Model_Versions = [Model, MFTADADELTACP, MFTADAGRADECP, MFTNADAMCP, MFTADAMAXCP,␣
      ↪MFTSGDCP]
```

```
[ ]: for i, model in enumerate(Model_Versions):
         print(f'Model no: {i}')
         test_loss, test_acc = model.evaluate(validation_batches, verbose=1)
         print(f'Validation accuracy:', test_acc)
         test_loss, test_acc = model.evaluate(testing_batches, verbose=1)
         print(f'Testing accuracy:', test_acc)
         print('-'*80+'\n')
```

```
Model no: 0
```

```
11/11 [==============================] - 1s 96ms/step - loss: 0.7314 - accuracy:
0.8500
Validation accuracy: 0.8500000238418579
11/11 [==============================] - 1s 106ms/step - loss: 0.7862 -
accuracy: 0.8271
Testing accuracy: 0.8271428346633911
--------------------------------------------------------------------------------

Model no: 1
11/11 [==============================] - 1s 104ms/step - loss: 1.3331 -
accuracy: 0.8729
Validation accuracy: 0.8728571534156799
11/11 [==============================] - 1s 99ms/step - loss: 1.9014 - accuracy:
0.8443
Testing accuracy: 0.8442857265472412
--------------------------------------------------------------------------------

Model no: 2
11/11 [==============================] - 1s 100ms/step - loss: 0.6877 -
accuracy: 0.8971
Validation accuracy: 0.8971428275108337
11/11 [==============================] - 1s 98ms/step - loss: 1.0381 - accuracy:
0.8629
Testing accuracy: 0.8628571629524231
--------------------------------------------------------------------------------

Model no: 3
11/11 [==============================] - 1s 98ms/step - loss: 1.0410 - accuracy:
0.8757
Validation accuracy: 0.8757143020629883
11/11 [==============================] - 1s 98ms/step - loss: 1.3044 - accuracy:
0.8471
Testing accuracy: 0.8471428751945496
--------------------------------------------------------------------------------

Model no: 4
11/11 [==============================] - 1s 95ms/step - loss: 1.7756 - accuracy:
0.8843
Validation accuracy: 0.8842856884002686
11/11 [==============================] - 1s 100ms/step - loss: 2.4966 -
accuracy: 0.8629
Testing accuracy: 0.8628571629524231
--------------------------------------------------------------------------------

Model no: 5
11/11 [==============================] - 1s 96ms/step - loss: 0.7016 - accuracy:
0.8471
Validation accuracy: 0.8471428751945496
```

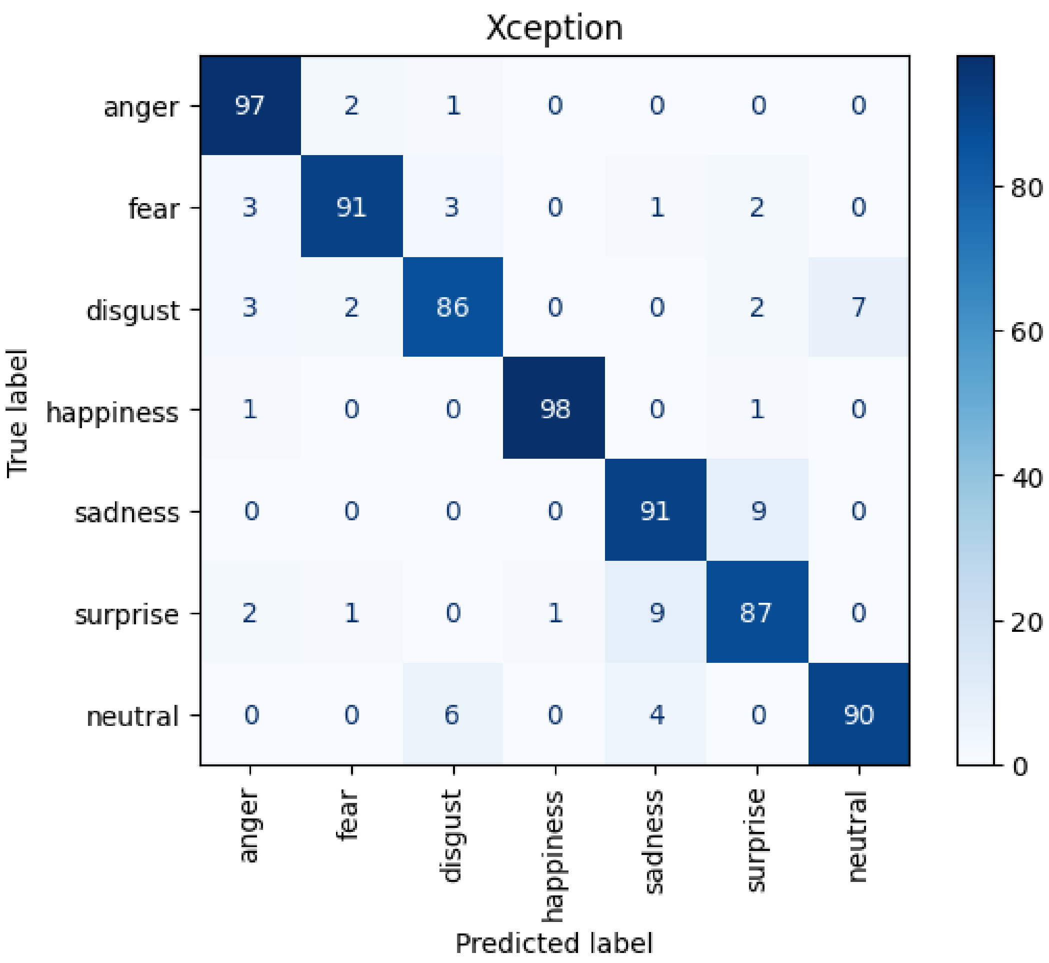
```
11/11 [==============================] - 1s 106ms/step - loss: 0.7160 -
accuracy: 0.8443
Testing accuracy: 0.8442857265472412
--------------------------------------------------------------------------------
```

5 Section 4: Evaluation

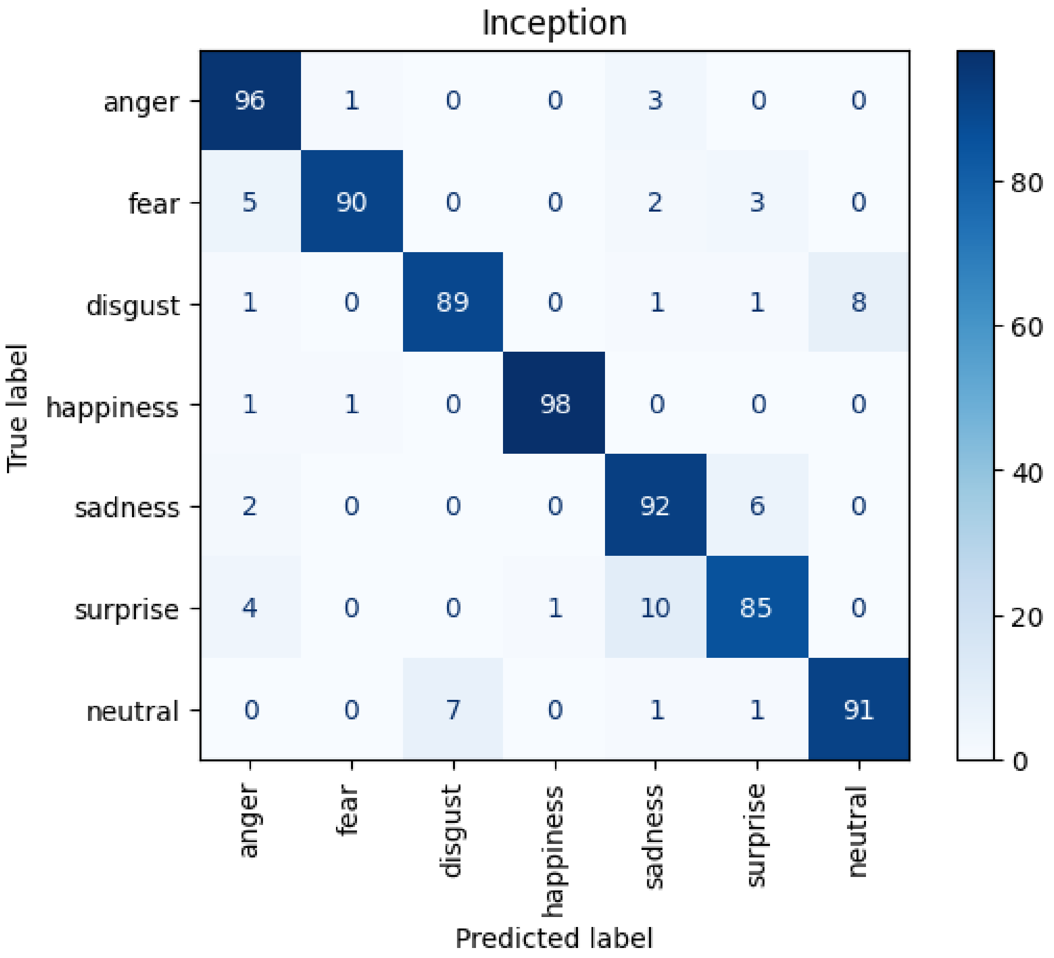
```
[ ]: Top_Models = {
         'Xception' : XFTNADAMCP,
         'Inception' : IFTNADAMCP,
         'Model' : MFTADAGRADECP
     }
```

5.1 4.1 Confusion Matrix

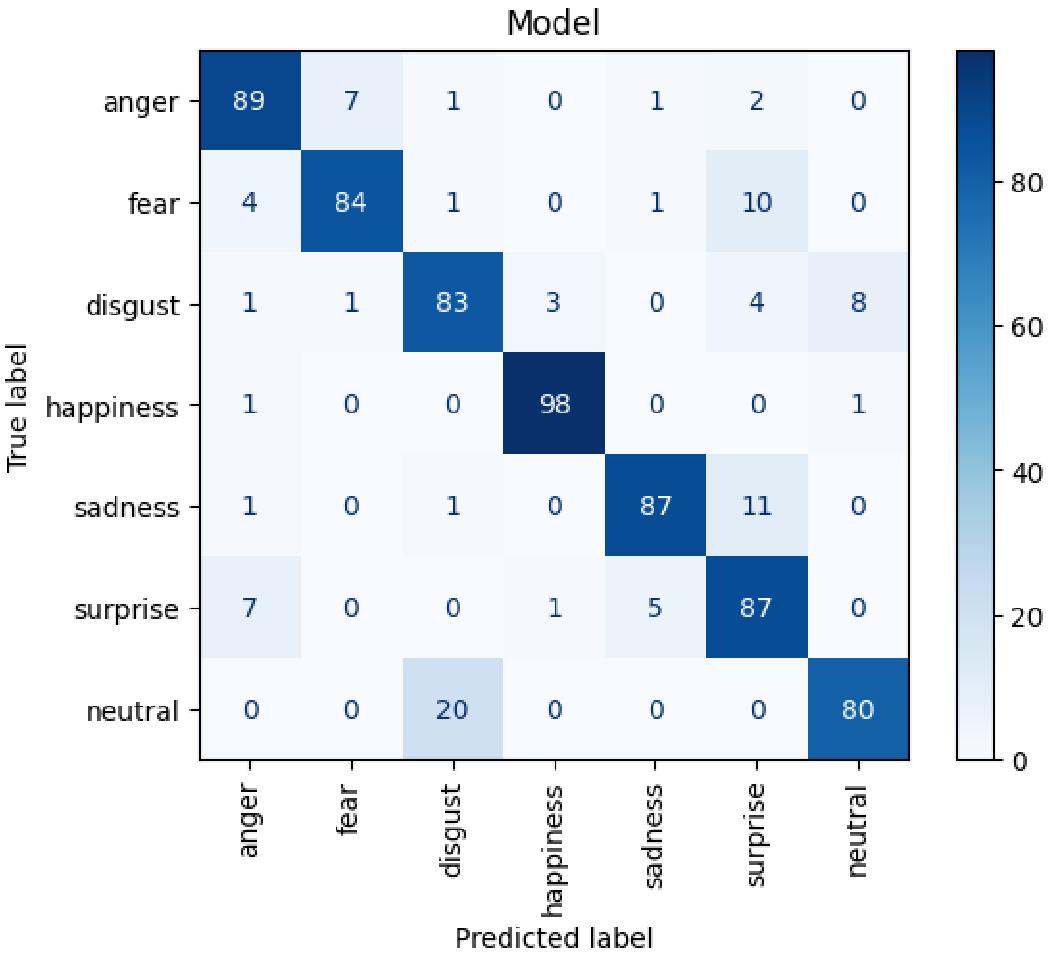
```
[ ]: for name, model in Top_Models.items():
         predictions = model.predict(x=testing_batches, verbose=0)
         np.round(predictions)
         cm = confusion_matrix(testing_batches.classes, np.argmax(predictions,␣
      ↪axis=1))
         class_labels = ['anger', 'fear', 'disgust', 'happiness', 'sadness',␣
      ↪'surprise', 'neutral']
         cm_display = ConfusionMatrixDisplay(confusion_matrix=cm,␣
      ↪display_labels=class_labels)
         cm_display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
         plt.title(name)
         plt.show()
         y_true = testing_batches.classes
         y_pred = np.argmax(predictions, axis=1)
         report = classification_report(y_true, y_pred, target_names=class_labels)
         print('-'*25 + name + '-'*25 + '\n')
         print(report)
```



```
-------------------------Xception-------------------------

              precision    recall  f1-score   support

       anger       0.92      0.97      0.94       100
        fear       0.95      0.91      0.93       100
     disgust       0.90      0.86      0.88       100
   happiness       0.99      0.98      0.98       100
     sadness       0.87      0.91      0.89       100
    surprise       0.86      0.87      0.87       100
     neutral       0.93      0.90      0.91       100

    accuracy                           0.91       700
   macro avg       0.91      0.91      0.91       700
weighted avg       0.91      0.91      0.91       700
```



```
-------------------------Inception-------------------------

              precision    recall  f1-score   support

       anger       0.88      0.96      0.92       100
        fear       0.98      0.90      0.94       100
     disgust       0.93      0.89      0.91       100
   happiness       0.99      0.98      0.98       100
     sadness       0.84      0.92      0.88       100
    surprise       0.89      0.85      0.87       100
     neutral       0.92      0.91      0.91       100

    accuracy                           0.92       700
   macro avg       0.92      0.92      0.92       700
weighted avg       0.92      0.92      0.92       700
```



```
-------------------------Model-------------------------

              precision    recall  f1-score   support

       anger       0.86      0.89      0.88       100
        fear       0.91      0.84      0.87       100
     disgust       0.78      0.83      0.81       100
   happiness       0.96      0.98      0.97       100
     sadness       0.93      0.87      0.90       100
    surprise       0.76      0.87      0.81       100
     neutral       0.90      0.80      0.85       100

    accuracy                           0.87       700
   macro avg       0.87      0.87      0.87       700
weighted avg       0.87      0.87      0.87       700
```

6 Section 5: Convert to Report

```
[ ]: !apt-get install texlive texlive-xetex texlive-latex-extra pandoc
     !pip install pypandoc
```

```
[ ]: !jupyter nbconvert --to PDF "/content/drive/MyDrive/
       ↪Muhammad_Ali_Thesis_3059828_FER/Machine_Learning_Thesis_FER.ipynb"
```

```
[ ]:
```