Facial Emotion Detection

Executive summary

The goal of this project is to build an artificial neural network that is capable of detecting human emotion through the analysis of human facial images.

It is assumed that this modeling tool will be used in concert with other biometric forms of behavorial analysis.

Multiple neural network architectures were tested in order to analyze human faces, including three transfer learning architectures and three scratch-built architectures consisting of four, five, and six blocks. The best performing model was a five convolutional block sequential network, using an Adam optimizer and relu activations for each convolutional block. This model will also be referred to as Model 2.

The images used to train the model were 48x48 pixel grayscale files, all subjectively labeled either "neutral", "sad", "happy", or "surprised". The number of incidences in each class were roughly even, except for a slightly lower incidence of images classed as "surprised". The imbalance in counts did not affect the final results of the training, partially due to the use of a data generator that was used to augment and multiply all of the classes.

In the end, the best model (a Convolutional Neural Network with five (5) convolutional blocks; Model 2) was capable of accurately identifying human emotions around 77% of the time. MODEL TWO (2) WAS CHOSEN BECAUSE IT PERFORMED BETTER BY PROVIDING THE BEST DISTINCTION BETWEEN CLASSES, that is, it had the highest accuracy and F1 scores for a higher number of classes than any of the other models. Model Two (2) performed as expected; and, upon close inspection of the labelled training data, peformance is roughly on par with human classification of facial emotion images. Most inaccurate predictions were a result of ambiguous labeling of sad and neutral data. Errors of either type (I or II) in classifying the "happy" and "surprised" classes were acceptably low.

The results of this project were very encouraging - while Model2 may seem to have low accuracy overall, it is clear that it could be applied with high levels of success when implemented properly for certain applications. Any situation that requires the positive identification of "happy" or "surprised" states, or anything situation that needs to exclude "happy" or "surprised" states from a set could find use of this model.

There are some concerns about the modeling process:

- Hyperparameters were tuned using the Keras Tuner; and in the interest of time, a small number of epochs were specified. Most of these tuners optimized validation accuracy @ about four (4) epochs. A close inspection of the resultant models, however, reveals that validation accuracy oscillated somewhat wildly over the first 10-20 epochs, so four (4) epochs likely did not return the best hyperparameter settings and a more refined tuning method might yield better accuracy results.

- The tuned learning rates were very small, so the model learned slowly. The cost of running the model on the V100 GPU was roughly 20-30 minutes, or $1 in machine time and roughly 40usd in wages (if the model is tended very closely). With future models I would try conservatively (and judiciously) adjusting the tuned learning rate to increase computing speeds.

  Next steps:

  - **If a distinction between the "sad" and "neutral" classes is not needed for any business application** , the two classes could simply be combined to dramatically increase the accuracy metrics of the model. For one-on-one human-robot interaction, *to err is human*. Take advantage of the ambiguity in classes to simulate human behavior. If there is ambiguity in interactions, it is possible to get more information about latent or unclear emotions simply by engaging in conversation, and this is actually probably preferred by humans (as opposed to being subject to inescapable powers of digital analysis to extract thoughts, ha ha).
  - **If a distinction between the "sad" and "neutral" classes is needed:**
    - **The model is adaptable, and the class sets can be somewhat cleaned as necessary.** For immediate use, if there is a **clear vision** regarding how the "sad" class will be used, then it may be helpful to immediately review all of the data in the "sad" folders and delete any that do not satisfy business requirements. Likewise, the "neutral" folder will need to be reviewed as well for the application. For example, if only *"sad and crying" subjects/"dramatically sad"* subjects qualify as "sad", then identify all *"lightly sad"/"slightly melancholy"* subjects and exclude them from training using the subroutine in the code. No "dramatically sad" subjects were observed in the "neutral" sets.
    - In time, if there are additional images to train a new model, new classes can be and should be added to capture other emotions and at this time classes for the mild to moderate cases of each class type can be added and correctly sorted and labeled. Other classes could possibly be named: "melancholy", "bored", "angry", "confused", "mixed feelings", or "undeterminable."
  - **Identify all use cases in which it would be useful to identify subjects of each class.** For example, facial emotion analysis could be used to determine customer engagement or sentiment.

  Time and monetary investment required:

  - for an immediate data cleaning and review: two to three persons @ three (3) hours per person. At an average of 75usd /hour, the cost is 450usd-675usd.
  - for ongoing review and updating: roughly two hours to alter and rerun the code for the specific model. Computational cost is roughly 10usd and total time required by the engineer two (2) hours. At 75usd/hour this is 150usd. The cost of an immediate review

of the model is roughly 760usd. The cost of ongoing review of comparable additions to the training data set would be the same, and it is recommended that updates occur annually at **610usd-830usd per year**.

The cost of the model would be offset by increased efficiency in customer targeting. For the retail industry or for a sales and marketing department, implementing the model would reduce costs of customer surveys, focus groups, and possibly negotiations.

Key challenges and risks: Data privacy, access/trust/perception, policy, ethics, and data security

The key challenges and costs are not from maintaining and running the model. As I hinted above, the public's imagination about AI and what it can do will present some challenges regarding the widespread use of automated technology that can individually profile individuals and extract (perceived) dynamic personal information from them merely by using facial expressions.

Clearly, images of faces are not hidden and people are not concerned that their faces can be seen, because people interact with one another in public on a daily basis. Allowing access through one-on-one extemporaneous conversation with a human being (that is not recorded or subject to repetitive, on-demand analysis) differs, however, from being scrutinized (possibly en-mass) by trained technologies to uncover and possibly exploit every facet of an individual's essence and identity.

Data about individuals will need to be stored responsibly, and there will need to be uncompromising levels of security to protect individuals sensitive and private data.

Data will also need to be used ethically in order to maintain trust and earn continued access to images.

Problem Definition

The context: Why is this problem important to solve?

Recent research has determined that communication is 93% non-verbal (not defined by literal words).

see: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6127604/

Furthermore, 55% of communication is visual, so in order to algorithmically interpret unprompted human communication, it is necessary to examine the many facets of visual and non-verbal communication.

This project and the subsequent model addresses the algorithmic interpretation of facial emotions. Facial detection using neural networks could further the advancement of artificial intelligence and can be used in a number of applications, for example:

- Accurate artificial interpretation of facial emotions could be used to develop more emotionally intelligent behavior in AI technologies, for example, voice assistants with cameras could use input from users' facial emotions as feedback. These assistants/robots could in turn take appropriate actions based on non-verbal responses and cues.

- Facial emotion detection technology could be applied to algorithms used for security purposes and combined with other biometric measurements to more accurately detect security threats. The facial emotion readings could be used in concert with other biometrics to analyze sentiments of participants in crowded events and could even possibly be used to deny entry into venues and buildings.
- Refined facial emotion detection technology could also be used to provide access to non-verbal communication to those who are not medically able to visually process facial emotions on their own, for the algorithms can be used for example, to help blind patients interpret emotions of individuals within a room or to help some users on the Autism Spectrum Disorder interpret non-verbal communication of multiple surrounding individuals.

The objectives: What is the intended goal?

The goal of this project is to build and train a neural network that can accurately identify the facial emotions of humans using 48x48 pixel, mid-to-high quality black and white images. It is assumed that the results of this emotion detection model will be used in concert with multiple biometric measurements and that inferences will be made on the basis of input from multiple available techniques.

Human emotion detection is partially possible through visual analysis and examination, however, facial emotion cues are not always reliable or a definitive measurement of facial emotion. As referenced above, it is often necessary to use other methods of communication and inspection to confirm a subject's specific emotional state. Other methods of ascertaining emotion include biometric analysis and verbal interaction, however, in many cases (especially in the circumstances where there are many people) other methods of interaction can be cost-prohibitive, time-consuming, or impractical by other standards. In cases like these, it would be helpful to be able to use facial emotion detection as a less invasive, lower-cost first layer of analysis that could identify individuals that should be subject to deeper levels of examination.

The key questions: What are the key questions that need to be answered?
What model works best, and how well does it work? We will examine numerous existing state-of-the art neural network models, as well as build our own to determine which is the best model architecture for Facial Emotion Detection. Ultimately, we will use a tool known as a classification report to provide us with a statistics that detail each model's accuracy, level of error, and error type. This information will be used to ascertain whether our models achieve an acceptable level of error and accuracy in facial emotion detection.
Which facial features are key to understanding facial emotion? We will also delve into determining which key facial features accurately relay facial emotion, and how to best clean and classify training data when building models for various applications.

The problem formulation: What are we trying to solve using data science?

It is assumed that this modeling tool will be used in concert with other biometric forms of behavorial analysis. For most applications, like crowd management, law enforcement, medicine, etc... a model that minimizes false negatives to a point where the model provides a good basis - or jumping off point - of nominating subjects of interest for further examination would be best. Specifically, the model must make it feasible to interact on a one-to-one basis with each flagged subject.
False positives should be minimized to the point where it would be reasonable to expect that any instance flagged by this application/model is worth the cost of interaction. The recall of a model must also be high enough to maintain the credibility of the tool. The level of accuracy, recall, and precision necessary for emotionally intelligent capabilities that would advance real-time robot to human interaction would need to be on par with human emotion detection, and when used with the right type of other information (such as verbal cues and conversational context). Specific reactions would need to give consideration and attention to error levels and their specific types for each class.

About the dataset The data set consists of 3 folders, i.e., 'test', 'train', and 'validation'. Each of these folders has four subfolders: **‘happy’**: Images of people who have happy facial expressions.
‘sad’: Images of people with sad or upset facial expressions.
‘surprise’: Images of people who have shocked or surprised facial expressions.
‘neutral’: Images of people showing no prominent emotion in their facial expression at all.

While most of the images in the dataset are classified correctly, some problems have been identified. Problems are of multiple types:

- Some of images are clearly not classified correctly, for example laughing smiling faces have been placed in the happy folder, and vice-versa
- Some of the images *may* be classified incorrectly or may be subject to wide interpretation: the neutral class of images presents us with a problem because there is no clear level for how disinterested a face must look in order to be classified as "neutral" in lieu of "sad" or "happy". Our models' results reflect this.
- Some of the images are clearly not human faces, for example, part of the data appears to be scraped from the internet, and some of the images are of "file not found" icons and and things of the like.

Mounting the Drive

NOTE: Please use Google Colab from your browser for this notebook. **Google.colab is NOT a library that can be downloaded locally on your device.**

In [1]:
```
# Mounting the drive
from google.colab import drive
```

```
drive.mount('/content/drive')
```

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

Importing the Libraries

In [2]:

```
!pip install -U efficientnet #install efficient net for efficient net project module
!pip install keras-tuner -q #install keras-tuner to set up hyperparameters for the cnn

#import packages

import glob #to read files and folders from drives
import os # to access files from user drives
import zipfile # to open zipped files
import cv2 # for computer vision to load and process images
import pandas as pd #to read files
import numpy as np # to use numpy arrays
import matplotlib.pyplot as plt # to graph and visualize data
import seaborn as sns # to graph data and visualize data
import tensorflow as tf # to train digit recognition training data
import random #to set random seed
import keras_tuner #to set up hyperparameters for the cnn models

# Importing Deep Learning Libraries

from tensorflow.keras.preprocessing.image import load_img, img_to_array  #image prepro
from tensorflow.keras.preprocessing.image import ImageDataGenerator #transform images
from keras import applications #need this to import transfer learning architectures
from tensorflow.keras import backend  # to clear session history
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D, Flatten, C
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.optimizers.legacy import Adamax # import legacy optimizers if th
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau #to impl
```

```
Collecting efficientnet
  Downloading efficientnet-1.1.1-py3-none-any.whl (18 kB)
Collecting keras-applications<=1.0.8,>=1.0.7 (from efficientnet)
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.7/50.7 kB 3.0 MB/s eta 0:00:00
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-package
s (from efficientnet) (0.19.3)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-package
s (from keras-applications<=1.0.8,>=1.0.7->efficientnet) (1.23.5)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from
keras-applications<=1.0.8,>=1.0.7->efficientnet) (3.9.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-package
s (from scikit-image->efficientnet) (1.11.3)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packag
es (from scikit-image->efficientnet) (3.1)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/local/li
b/python3.10/dist-packages (from scikit-image->efficientnet) (9.4.0)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packa
ges (from scikit-image->efficientnet) (2.31.5)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->efficientnet) (2023.9.26)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-pa
ckages (from scikit-image->efficientnet) (1.4.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pack
ages (from scikit-image->efficientnet) (23.2)
Installing collected packages: keras-applications, efficientnet
Successfully installed efficientnet-1.1.1 keras-applications-1.0.8
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.5/129.5 kB 3.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 950.8/950.8 kB 9.2 MB/s eta 0:00:00
Using TensorFlow backend
```

Let us load and unzip the data

Note:

- You must download the dataset from the link provided on Olympus and upload the same on your Google drive before executing the code in the next cell.
- In case of any error, please make sure that the path of the file is correct as the path may be different for you.

In [3]:
```
# Extract from zip using zipfile pkg
with zipfile.ZipFile('/content/drive/MyDrive/Facial_emotion_images.zip', 'r') as sourc
    source.extractall()
```

In [4]:
```
#images destination
#folders test, train, data each holding expressions happy, neutral, sad, surprise and
extract_to="Facial_emotion_images"
```

In [5]:
```
#this picks up all of the top level folders in the directory
folders = glob.glob("./Facial_emotion_images/*")
```

In [6]:
```
#checking the number of folders in the directory, and printing the name of the folders
print("total number of folders:", len(folders))
print("Showing all", len(folders), "folders...")
folders[:len(folders)]
```

```
total number of folders: 3
Showing all 3 folders...
```

Out[6]:
```
['./Facial_emotion_images/train',
 './Facial_emotion_images/validation',
 './Facial_emotion_images/test']
```

There are three folders in the zipped file named test, validation, and train. Here are the contents of the folders.

In [7]:
```
#return the different folders name with filepath
test_contents,validation_contents, train_contents = glob.glob("./Facial_emotion_images
```

In [8]:
```
#prints the different folder names with filepath
print("There are", len(test_contents), "items in test_contents:", test_contents[:len(t
print("There are", len(validation_contents), "items in validation_contents:", test_cor
print("There are", len(train_contents), "items in train_contents:", test_contents[:ler
```

```
There are 4 items in test_contents: ['./Facial_emotion_images/test/sad', './Facial_em
otion_images/test/neutral', './Facial_emotion_images/test/surprise', './Facial_emotio
n_images/test/happy']
There are 4 items in validation_contents: ['./Facial_emotion_images/test/sad', './Fac
ial_emotion_images/test/neutral', './Facial_emotion_images/test/surprise', './Facial_
emotion_images/test/happy']
There are 4 items in train_contents: ['./Facial_emotion_images/test/sad', './Facial_e
motion_images/test/neutral', './Facial_emotion_images/test/surprise', './Facial_emoti
on_images/test/happy']
```

Each folder contains four subfolders. The names of the subfolders appear to be our subclasses (neutral, surprise, happy, and sad).

In [9]:
```
#manually naming the subclasses based on folder title and filepath
classes= ("neutral", "surprise", "happy", "sad" )
groups= ("test", "validation", "train")
where_is = "./Facial_emotion_images/"
```

In [10]:
```
#creating a tuple to hold the names of all of the classification folders
sourcefile_path=[]
for i in range (len(classes)):
  sourcefile_path.append(where_is+"train"+"/"+classes[i]+"/*",),
print(sourcefile_path)
```

```
['./Facial_emotion_images/train/neutral/*', './Facial_emotion_images/train/surprise/
*', './Facial_emotion_images/train/happy/*', './Facial_emotion_images/train/sad/*']
```

Let's check the content of the subfolders to confirm that they hold our image files.

In [11]:
```
#Create a dictionary with the number of images in each class to view the distribution
class_sizes=[]
for i in range (len(sourcefile_path)):
  class_sizes.append({'emotion': classes[i], 'count': len(glob.glob(sourcefile_path[i]
print (class_sizes)
```

```
[{'emotion': 'neutral', 'count': 3978}, {'emotion': 'surprise', 'count': 3173}, {'emo
tion': 'happy', 'count': 3976}, {'emotion': 'sad', 'count': 3982}]
```

In [12]:
```
#checking the code above to make sure the filepaths were created, and how each emotion
face=sourcefile_path[2]
print(face)
```

./Facial_emotion_images/train/happy/*

For our training set, there is a relatively small number of images in each class, it appears that we might need to generate extra data.

In [13]:
```
#check the shape of our training data by using computer vision (cv2)
test1=cv2.imread(glob.glob(face)[2])
print(type(test1))
print(test1.shape)
```

```
<class 'numpy.ndarray'>
(48, 48, 3)
```

Our images are 48 pixels by 48 pixels with 3 color channels.

Visualizing our Classes

Let's look at our classes.

Write down your observation for each class. What do you think can be a unique feature of each emotion, that separates it from the remaining classes?

General observations based on visual inspection all of the images. Most of the images are allocated to training the model, which is fine, because there is a limited number of images to begin with. Consequently, our predominant data integrity issues can be found in the training set. In each of the folders, we find that some images of our classified incorrectly.

- We often find instances that are not faces, but that are only text or icons.
- We also find instances where photos of faces are placed in the wrong class.

We will delete these instances before checking the distribution of the training classes.

Happy

In [14]:
```
cols=4 #plot size, cols
rows=3 #plot size, rows
ts = test1.shape[0] #define input size
```

In [15]:
```
import random

#set random seed
np.random.seed(7)
random.seed(7)
tf.random.set_seed(7)
```

In [16]:
```
#happy class is source file path 2
#create plot of sample faces below
face=sourcefile_path[2]
plt.figure(figsize= (12,8))
for i in range(1,cols*rows+1,1):
    plt.subplot(rows,cols,i)
    img=load_img(glob.glob(face)[i], target_size=(ts,ts))
```

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

Observations and Insights:

GENERAL

Generally, subjects are smiling, with causes roundness in the upper cheeks and a general widening of the mouth area. The classical pronounced "U" shape can be seen particularly on the lower lips. There are also well-defined creases extending from the sides of the nose down to the corners of the mouth. Chins also tend to point as well.

MOUTH

In smilling (happy) and frowning (sad) photos, the corners of the mouth are not lined up with the pupils of the eyes, while in a neutral photo, they should be. For a closed smile, relative to the center point of the top lip (or relative to the center of the whole mouth when open), the two distances from the corners of the mouth to the center of the top lip are greater than they would be if the mouth was held in a flat neutral position.

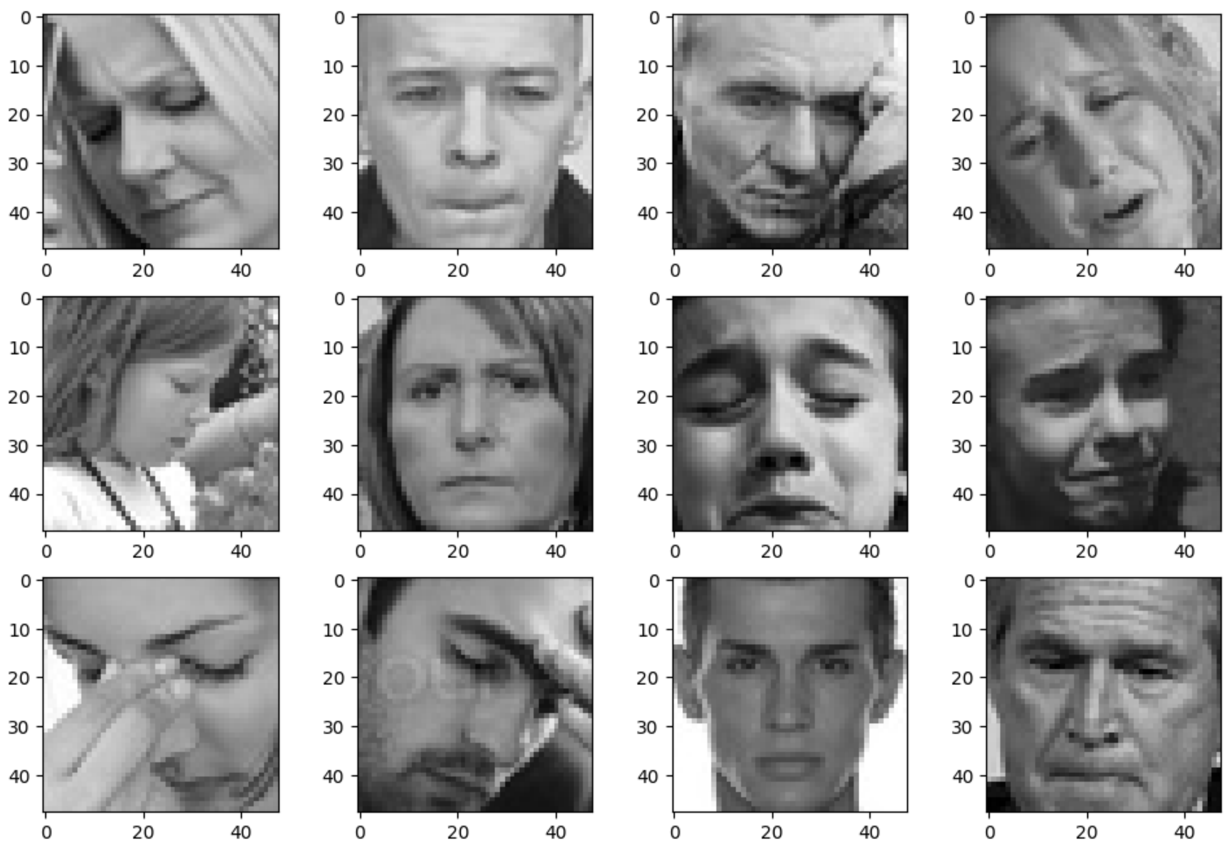
If a typical x-y cartesian axis was placed on the mouth, where y is along the vertical axis, and x is along the horizontal axis, the position of the corners of the mouth would change based on the emotion or expression. In general, ***happy expressions could be indicated with lifted mouth corners that lie above the x-axis***, and sad expressions can be approximated by corners that lie below the x-axis.

Statistically, the mouth vectors could be grouped and roughly clustered to form one parameter for classification.

Refer to any basic drawing course to further research basic facial proportions. A quick resource can be found here: https://www.thedrawingsource.com/proportions-of-the-face.html

Sad

In [17]:
```
#sad class is source file path 3
#create plot of sample faces below
face=sourcefile_path[3]
plt.figure(figsize= (12,8))
for i in range(1,cols*rows+1,1):
    plt.subplot(rows,cols,i)
    img=load_img(glob.glob(face)[i], target_size=(ts,ts))
    plt.imshow(img)
plt.show(img)
```



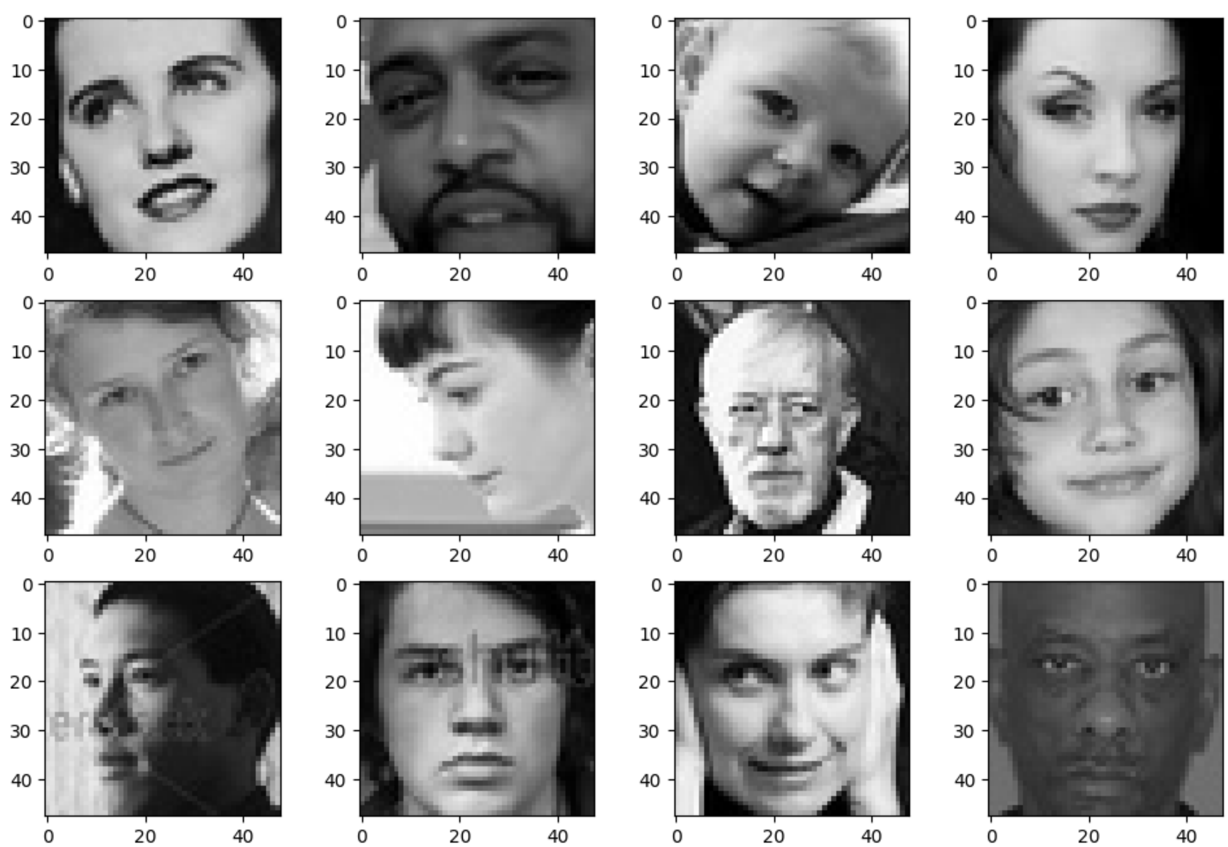
Observations and Insights:__Some of our images are apparently watermarked. The system will need to learn to filter these watermarks out of the analysis. Generally eyes are closed, downturned at the outer corner, or squinted. Lips are often pursed. The person may be using another body part to hide the parts of the face that closest to the eyes.

These training images in the sad class are very diverse and contain tears, hands, and many different facial positions and expressions, so this class might be tougher to classify due to its diversity and complexity. Some of the images could easily be interpreted as neutral images, and this could prove challenging for the model to differentiate.

Neutral

In [18]:

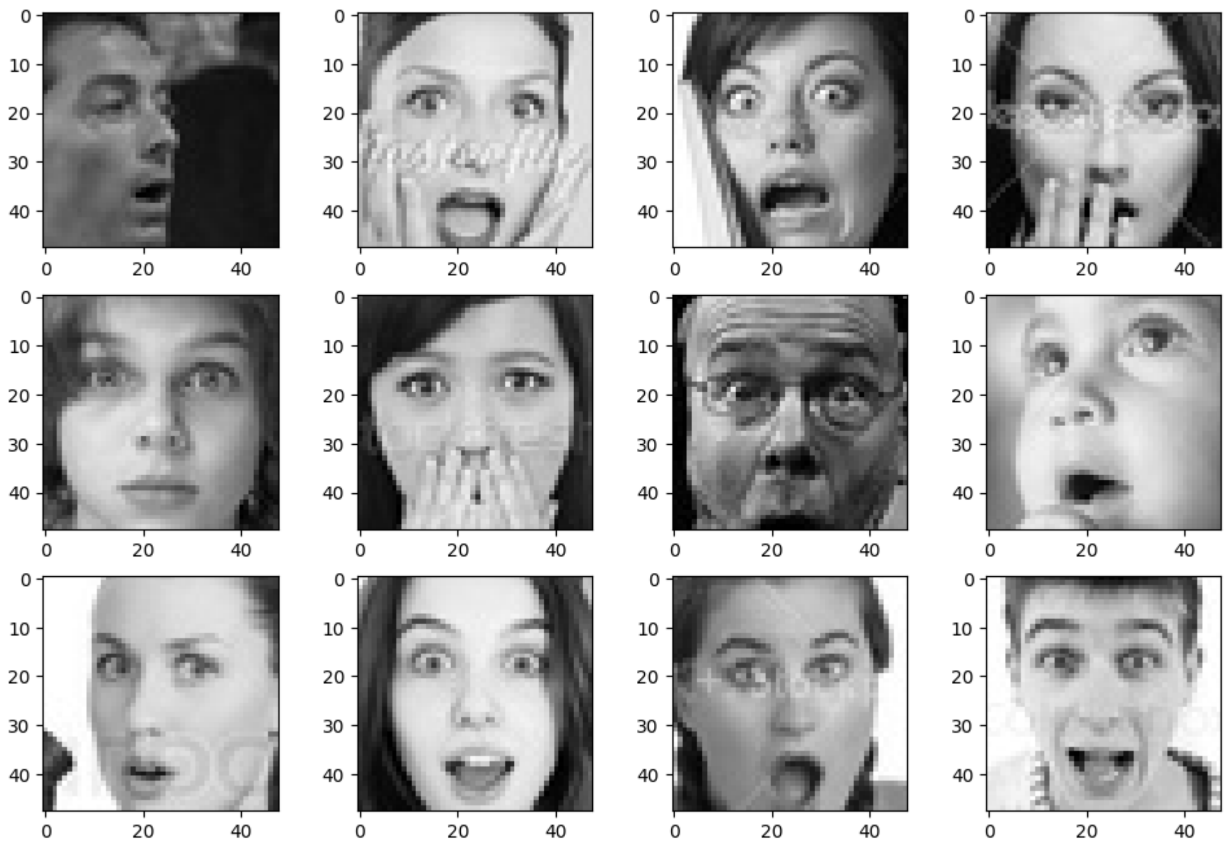
```
#neutral class is source file path 0
#create plot of sample faces below
face=sourcefile_path[0]
plt.figure(figsize= (12,8))
for i in range(1,cols*rows+1,1):
    plt.subplot(rows,cols,i)
    img=load_img(glob.glob(face)[i], target_size=(ts,ts))
    plt.imshow(img)
plt.show(img)
```



Observations and Insights: Mouth and lips are generally closed (there appears to be at least one image where that is not the case). Brows also tend to be relaxed.

Surprised

In [19]:
```
#surprised class is source file path 1
#create plot of sample faces below
face=sourcefile_path[1]
plt.figure(figsize= (12,8))
for i in range(1,cols*rows+1,1):
    plt.subplot(rows,cols,i)
    img=load_img(glob.glob(face)[i], target_size=(ts,ts))
    plt.imshow(img)
plt.show(img)
```



Checking Distribution of Classes

Observations and Insights: Eyebrows have a pronounced curve downward, mouth is open/ or lips are shaped like the letter O, and eyes are big.

Facial proportions are stretched due to the exaggerated muscle movements that typically accompany a surprised expression.

It's probable that the ratio of the distances from the center point of the chin to base of the nose (or the center point of the eyes) plays are role in classifying these images.

We also see that some of the files are mislabeled as faces (when they are pictures of "file not found text", emoticons, or other items). Other instances of bad data are faces that are obviously misclassified.

Let's clean and delete the mislabeled and misclassified files from the training set and then graph the distribution of the training classes.

In [20]:
```
#DELETING BAD DATA
#check current working directory to make sure that we're in the right location on Goog
os.getcwd()
```

Out[20]:
```
'/content'
```

In [21]:
```
#concatenate file paths: train using a loop
trainfile_path=[]
for i in range (len(classes)):
  trainfile_path.append(where_is+"train"+"/"+classes[i]+"/",),
print(trainfile_path)
```

```
['./Facial_emotion_images/train/neutral/', './Facial_emotion_images/train/surprise/',
'./Facial_emotion_images/train/happy/', './Facial_emotion_images/train/sad/']
```

In [22]:
```
#concatenate file paths: validation using a loop
valfile_path=[]
for i in range (len(classes)):
  valfile_path.append(where_is+"validation"+"/"+classes[i]+"/",),
print(valfile_path)
```

```
['./Facial_emotion_images/validation/neutral/', './Facial_emotion_images/validation/s
urprise/', './Facial_emotion_images/validation/happy/', './Facial_emotion_images/vali
dation/sad/']
```

In [23]:
```
#concatenate file paths: test using a loop
testfile_path=[]
for i in range (len(classes)):
  testfile_path.append(where_is+"test"+"/"+classes[i]+"/",),
print(testfile_path)
```

```
['./Facial_emotion_images/test/neutral/', './Facial_emotion_images/test/surprise/',
'./Facial_emotion_images/test/happy/', './Facial_emotion_images/test/sad/']
```

In [24]:
```
#check if a file is found
if os.path.exists('/content/Facial_emotion_images/train/surprise/26557.jpg'):
#if os.path.exists(os.path.join(trainfile_path[2],trainhappydel[1])):
  print('The file exists.')
else:
  print('The file does not exist.')
```

```
The file exists.
```

Here we delete the misclassified and mislabeled files from the training set and pass the counts for each class to a dataframe

In [25]:
```
#deleting files that are misclassified based on visual/manual inspection
#classes in order
#neutral 0
#surprise 1
#happy2
#sad3

#mount drive to find the
drive.mount("/content/drive", force_remount=True)

classrun=["neutral", "surprise", "happy", "sad"] #define classes to run through
```

```
#define files to delete, training folder
trainhappydel=('24891.jpg','25603.jpg','26383.jpg','27241.jpg','27260.jpg','16170.jpg'
trainsaddel= ('29710.jpg','23596.jpg','22093.jpg','10860.jpg','23894.jpg','30705.jpg',
trainsurprisedel=('19238.jpg','26557.jpg')
trainneutraldel=('31127.jpg','11525.jpg','11600.jpg','11846.jpg','11864.jpg','11860.jp


#LOOP THROUGH AND DELETE FILES DEFINED ABOVE BY CLASS
#set indices, i and cl to zero
cl=0
i=0
for cl in range(len(classrun)): #run this loop 4 times, stepping through all elements
  file_path=("train"+classrun[cl]+"del")    #get the concatenated filepath
  for i in range(len(eval(file_path))):    #for each item in the list named by the str
    tup = eval(file_path)[i-1]                           #evaluate the string "concate
    cleanfile=os.path.join(trainfile_path[int(cl)],(tup)) #then join the directory nam
    if os.path.exists(cleanfile):                        #IFF the file is present, the
      os.remove(cleanfile)                               #delete that file, otherwise:
      print(f'deleted {cleanfile} at position {i} and class {cl}')          #print th
    i += 1                                                    #otherwise go to the nex
  cl +=1                                                      #and when you run out, g


#Define class sizes of test set and
class_sizes=[]
for i in range (len(sourcefile_path)):
  class_sizes.append({'emotion': classes[i], 'count': len(glob.glob(sourcefile_path[i]
print (class_sizes)
```

```
Mounted at /content/drive
deleted ./Facial_emotion_images/train/neutral/35469.jpg at position 0 and class 0
deleted ./Facial_emotion_images/train/neutral/31127.jpg at position 1 and class 0
deleted ./Facial_emotion_images/train/neutral/11525.jpg at position 2 and class 0
deleted ./Facial_emotion_images/train/neutral/11600.jpg at position 3 and class 0
deleted ./Facial_emotion_images/train/neutral/11846.jpg at position 4 and class 0
deleted ./Facial_emotion_images/train/neutral/11864.jpg at position 5 and class 0
deleted ./Facial_emotion_images/train/neutral/11860.jpg at position 6 and class 0
deleted ./Facial_emotion_images/train/neutral/11985.jpg at position 7 and class 0
deleted ./Facial_emotion_images/train/neutral/13323.jpg at position 8 and class 0
deleted ./Facial_emotion_images/train/neutral/15144.jpg at position 9 and class 0
deleted ./Facial_emotion_images/train/neutral/18062.jpg at position 10 and class 0
deleted ./Facial_emotion_images/train/neutral/31059.jpg at position 11 and class 0
deleted ./Facial_emotion_images/train/neutral/19713.jpg at position 12 and class 0
deleted ./Facial_emotion_images/train/neutral/19632.jpg at position 13 and class 0
deleted ./Facial_emotion_images/train/neutral/22246.jpg at position 14 and class 0
deleted ./Facial_emotion_images/train/neutral/22927.jpg at position 15 and class 0
deleted ./Facial_emotion_images/train/neutral/24734.jpg at position 16 and class 0
deleted ./Facial_emotion_images/train/neutral/26053.jpg at position 17 and class 0
deleted ./Facial_emotion_images/train/neutral/26380.jpg at position 18 and class 0
deleted ./Facial_emotion_images/train/neutral/26513.jpg at position 19 and class 0
deleted ./Facial_emotion_images/train/neutral/26897.jpg at position 20 and class 0
deleted ./Facial_emotion_images/train/neutral/30373.jpg at position 21 and class 0
deleted ./Facial_emotion_images/train/neutral/31745.jpg at position 22 and class 0
deleted ./Facial_emotion_images/train/neutral/31823.jpg at position 23 and class 0
deleted ./Facial_emotion_images/train/neutral/31960.jpg at position 24 and class 0
deleted ./Facial_emotion_images/train/neutral/31956.jpg at position 25 and class 0
deleted ./Facial_emotion_images/train/neutral/31974.jpg at position 26 and class 0
deleted ./Facial_emotion_images/train/neutral/31993.jpg at position 27 and class 0
deleted ./Facial_emotion_images/train/neutral/34334.jpg at position 28 and class 0
deleted ./Facial_emotion_images/train/neutral/35401.jpg at position 29 and class 0
deleted ./Facial_emotion_images/train/surprise/26557.jpg at position 0 and class 1
deleted ./Facial_emotion_images/train/surprise/19238.jpg at position 1 and class 1
deleted ./Facial_emotion_images/train/happy/21206.jpg at position 0 and class 2
deleted ./Facial_emotion_images/train/happy/24891.jpg at position 1 and class 2
deleted ./Facial_emotion_images/train/happy/25603.jpg at position 2 and class 2
deleted ./Facial_emotion_images/train/happy/26383.jpg at position 3 and class 2
deleted ./Facial_emotion_images/train/happy/27241.jpg at position 4 and class 2
deleted ./Facial_emotion_images/train/happy/27260.jpg at position 5 and class 2
deleted ./Facial_emotion_images/train/happy/16170.jpg at position 6 and class 2
deleted ./Facial_emotion_images/train/happy/16227.jpg at position 7 and class 2
deleted ./Facial_emotion_images/train/happy/16540.jpg at position 8 and class 2
deleted ./Facial_emotion_images/train/happy/17030.jpg at position 9 and class 2
deleted ./Facial_emotion_images/train/sad/25957.jpg at position 0 and class 3
deleted ./Facial_emotion_images/train/sad/29710.jpg at position 1 and class 3
deleted ./Facial_emotion_images/train/sad/23596.jpg at position 2 and class 3
deleted ./Facial_emotion_images/train/sad/22093.jpg at position 3 and class 3
deleted ./Facial_emotion_images/train/sad/10860.jpg at position 4 and class 3
deleted ./Facial_emotion_images/train/sad/23894.jpg at position 5 and class 3
deleted ./Facial_emotion_images/train/sad/30705.jpg at position 6 and class 3
deleted ./Facial_emotion_images/train/sad/32850.jpg at position 7 and class 3
[{'emotion': 'neutral', 'count': 3948}, {'emotion': 'surprise', 'count': 3171}, {'emo
tion': 'happy', 'count': 3966}, {'emotion': 'sad', 'count': 3974}]
```

In [26]:

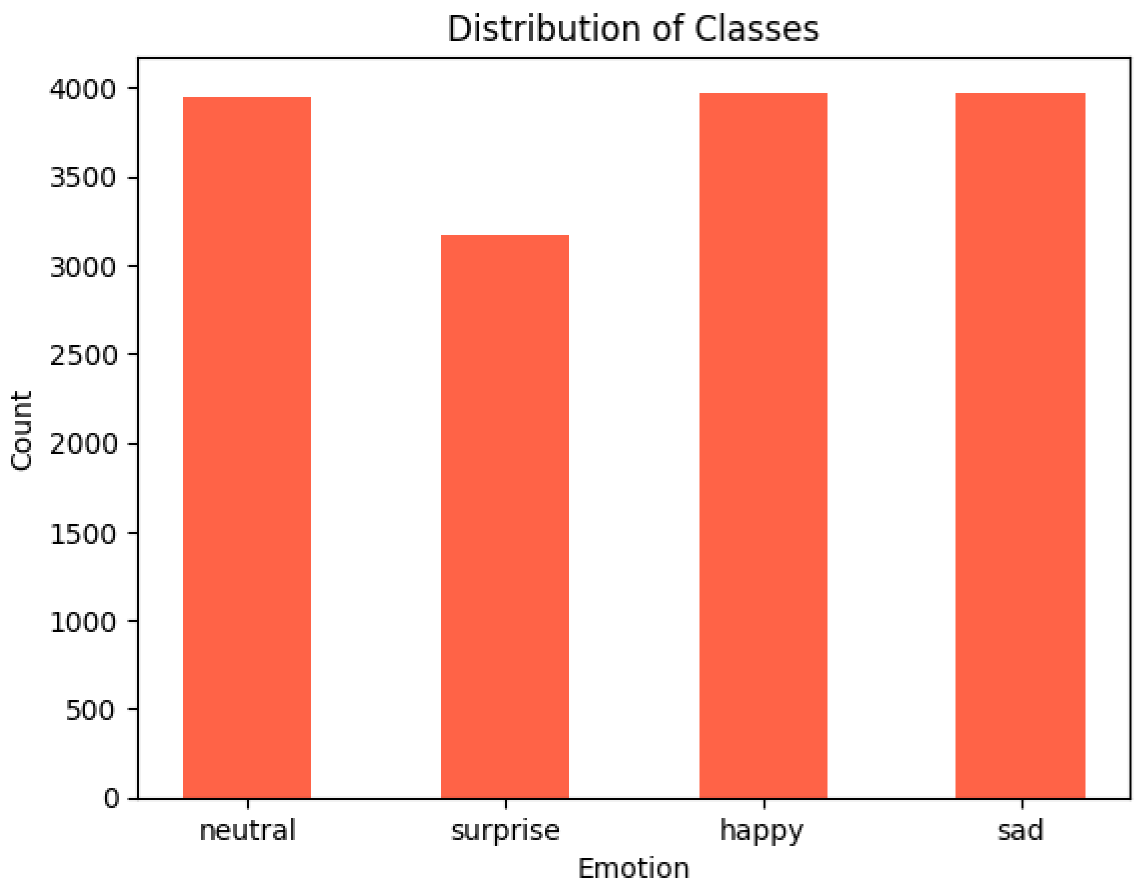
```
#convert the list of dictionaries into a dataframe, named df
df=pd.DataFrame(class_sizes)
#check dataframe structure
df.head()
```

Out[26]:

	emotion	count
0	neutral	3948
1	surprise	3171
2	happy	3966
3	sad	3974

In [27]:

```
#plot distribution of classes
plt.bar(df['emotion'], df['count'], width=0.5, color=('tomato'))#added color to change
plt.title('Distribution of Classes')
plt.xlabel('Emotion')
plt.ylabel('Count')
plt.show()
```



Observations and Insights:_Classes are unbalanced; "surprise" has the lowest number of instances; this would be unremarkable except for the fact that the counts remaining three classes are fairly equal. We will need to generate more images to even out the number of classes and also to increase the diversity of the training set

We will do this by using an image data generator after we clean the validation and test data. These data sets are a lot smaller than the training set and there are fewer data points to clean.

Cleaning Validation and Test Sets

In [28]:

```
#Cleaning validation set
#deleting files that are misclassified based on visual/manual inspection
#here are our classes and their indices
#neutral 0
#surprise 1
#happy2
#sad3


drive.mount("/content/drive", force_remount=True) #remount drive to find all files

#manual input of files to delete from the validation folder
valhappydel=('30981.jpg', 'none.jpg')
valsaddel= ('none.jpg', 'none.jpg')
valsurprisedel=('29557.jpg','35121.jpg')
valneutraldel=('12289.jpg','21817.jpg','32683.jpg')


classrun=["neutral", "surprise", "happy", "sad"] #define an index of classes to run th

#LOOP THROUGH AND DELETE FILES DEFINED ABOVE

#set indices, i and cl to zero
cl=0
i=0
for cl in range(len(classrun)):                     #run this loop 4 times, stepping t
  file_path=("val"+classrun[cl]+"del")                #get the concatenated filepath
  for i in range(len(eval(file_path))):                 #for each item in the list nam
    tup = eval(file_path)[i-1]                          #evaluate the string "concate
    cleanfile=os.path.join(valfile_path[int(cl)],(tup)) #then join the directory name
    if os.path.exists(cleanfile):                       #IFF the file is present, the
      os.remove(cleanfile)                              #delete that file, otherwise
      print(f'deleted {cleanfile} at position {i} and class {cl}')    #print th
    i += 1                                                   #otherwise go to the nex
  cl +=1                                                     #and when you run out, g
```

```
Mounted at /content/drive
deleted ./Facial_emotion_images/validation/neutral/32683.jpg at position 0 and class
0
deleted ./Facial_emotion_images/validation/neutral/12289.jpg at position 1 and class
0
deleted ./Facial_emotion_images/validation/neutral/21817.jpg at position 2 and class
0
deleted ./Facial_emotion_images/validation/surprise/35121.jpg at position 0 and class
1
deleted ./Facial_emotion_images/validation/surprise/29557.jpg at position 1 and class
1
deleted ./Facial_emotion_images/validation/happy/30981.jpg at position 1 and class 2
```

In [29]:

```
#Cleaning test set
#deleting files that are misclassified based on visual/manual inspection
#classes
#neutral 0
#surprise 1
#happy2
#sad3

drive.mount("/content/drive", force_remount=True) #remount drive to make sure we find
```

```
#define files to delete, test folder
testhappydel=('15838.jpg','6958.jpg')
testsaddel= ('none.jpg','none.jpg')
testsurprisedel=('none.jpg', 'none.jpg')
testneutraldel=('none.jpg', 'none.jpg')


classrun=["neutral", "surprise", "happy", "sad"] #define an index of classes to run th

#LOOP THROUGH AND DELETE FILES DEFINED ABOVE
#set indices, i and cl to zero
cl=0
i=0
for cl in range(len(classrun)): #run this loop 4 times, stepping through all elements
  file_path=("test"+classrun[cl]+"del")    #get the concatenated filepath
  for i in range(len(eval(file_path))):    #for each item in the list named by the str
    tup = eval(file_path)[i-1]                          #evaluate the string "concate
    cleanfile=os.path.join(trainfile_path[int(cl)],(tup)) #then join the directory nam
    if os.path.exists(cleanfile):                       #IFF the file is present, the
      os.remove(cleanfile)                              #delete that file, otherwise
      print(f'deleted {cleanfile} at position {i} and class {cl}')          #print th
    i += 1                                                   #otherwise go to the nex
  cl +=1                                                     #and when you run out, g
```

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

Think About It:

- Are the classes equally distributed? If not, do you think the imbalance is too high? Will it be a problem as we progress?
- Are there any Exploratory Data Analysis tasks that we can do here? Would they provide any meaningful insights?

Creating our Data Loaders

In this section, we are creating data loaders that we will use as inputs to our Neural Network.

You have two options for the color_mode. You can set it to color_mode = 'rgb' or color_mode = 'grayscale'. You will need to try out both and see for yourself which one gives better performance.

rotation_range is a value in degrees (0-180), a range within which to randomly rotate pictures

width_shift and height_shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally rescale is a value by which we will multiply the data before any other processing.

Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255 factor.

shear_range is for randomly applying shearing transformations zoom_range is for randomly zooming inside pictures

horizontal_flip is for randomly flipping half of the images horizontally --relevant when there are no assumptions of horizontal assymetry (e.g. real-world pictures).

fill_mode is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

In [30]:
```
# let's generate some training data employing the same methods used in the Rice Type a
# the data that the models use will be passed through this data augmentation process

color_is='grayscale'

batch_size  = 32


datagen_train  = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    height_shift_range=0.2,
    width_shift_range=0.2,
    brightness_range=(0.5,1.5),
    shear_range = 0.2,
    rotation_range= 60)

datagen_val = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    height_shift_range=0.2,
    width_shift_range=0.2,
    brightness_range=(0.5,1.5),
    shear_range = 0.2,
    rotation_range= 60)

datagen_test = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    height_shift_range=0.2,
    width_shift_range=0.2,
    brightness_range=(0.5,1.5),
    shear_range = 0.2,
    rotation_range= 60)

train_set = datagen_train.flow_from_directory(where_is+"train",
                                              target_size = (ts, ts),
                                              color_mode = color_is,
                                              batch_size = batch_size,
                                              seed = 5,
                                              class_mode = 'categorical',
                                              classes = classes,
                                              shuffle = True)

validation_set = datagen_val.flow_from_directory(where_is + 'validation/',
                                              target_size = (ts,ts),
                                              color_mode = color_is,
                                              batch_size = batch_size,
                                              seed = 5,
                                              class_mode = 'categorical',
```

```
                                                      classes = classes,
                                                      shuffle = True)

test_set = datagen_test.flow_from_directory(where_is + 'test/',
                                              target_size = (ts,ts),
                                              color_mode = color_is,
                                              batch_size = batch_size,
                                              seed = 5,
                                              class_mode = 'categorical', #in lieu of
                                              classes = classes,
                                              shuffle = True)
```

```
Found 15059 images belonging to 4 classes.
Found 4971 images belonging to 4 classes.
Found 128 images belonging to 4 classes.
```

In [31]:
```
print(test_set.class_indices)
print(test_set)
```

```
{'neutral': 0, 'surprise': 1, 'happy': 2, 'sad': 3}
<keras.src.preprocessing.image.DirectoryIterator object at 0x7bb8c486dcf0>
```

Base Model Building

Creating the Base Neural Network

In [32]:
```
# Clearing backend
from tensorflow.keras import backend
backend.clear_session()

#set random seed
np.random.seed(5)
random.seed(5)
tf.random.set_seed(5)
```

In [33]:
```
#set up a hyperparameter tuner
#https://keras.io/guides/keras_tuner/getting_started/

#add leaky-relu parameter alpha=0.1 to the list of activation functions that are able
from tensorflow.keras.utils import get_custom_objects
get_custom_objects().update({'leaky-relu': Activation(LeakyReLU(alpha=0.1))})

#define drop rate
drop=0.25

#define INPUT SHAPE, LAST CHANNEL 1 FOR GS, 3 FOR RGB
inputshape=(48,48,1)

#define possible activation parameters for tuning
activation1 = "leaky-relu"
activation2 = "ReLU"
activation3 = "selu"

#define loss function for compiling the model
loss_function = 'categorical_crossentropy'

def build_model(hp):
```

```
    activation=hp.Choice("activation", [activation1, activation2, activation3]) #tune th
    learning_rate = hp.Float("learning_rate", min_value=1e-4, max_value=1e-2, sampling='

  #build model 1
    model1 = Sequential()

  # First Convolutional Block
    units1=hp.Int("units1", min_value=32, max_value=512, step=32) #tune units to use for
    model1.add(Conv2D(units1, kernel_size=(3, 3), input_shape = (inputshape), padding =

  # Second Convolutional Block
    units2=hp.Int("units2", min_value=32, max_value=512, step=32) #tune units to use for
    model1.add(Conv2D(units2, kernel_size=(4, 4), padding = 'same', activation=activatio
  #Max Pooling
    model1.add(MaxPooling2D(2, 2))
  #Add a dropout layer
    model1.add(Dropout(drop))
  #Add a BatchNormalization layer
    model1.add(BatchNormalization())

  # Third Convolutional Block
    units3=hp.Int("units3", min_value=32, max_value=512, step=32) #tune units to use for
    model1.add(Conv2D(units3, kernel_size=(2, 2), padding = 'same', activation=activatio


  # Fourth Convolutional Block
    units4=hp.Int("units4", min_value=32, max_value=512, step=32) #tune units to use for
    model1.add(Conv2D(units4, kernel_size=(3, 3), padding = 'same', activation=activatio
  #Max Pooling
    model1.add(MaxPooling2D(2, 2))
  #Add a dropout layer
    model1.add(Dropout(drop))
  #Add a BatchNormalization layer
    model1.add(BatchNormalization())

  #flatten
    model1.add(Flatten())

  # First fully Connected Block
  #default activation is relu
    unitsfc1=hp.Int("unitsfc1", min_value=32, max_value=512, step=32) #tune units to use
    model1.add(Dense(unitsfc1))
  #Add a dropout layer
    model1.add(Dropout(drop))

  #model1.add(Flatten())

  # Second fully Connected Block
    unitsfc2=hp.Int("unitsfc2", min_value=32, max_value=512, step=32) #tune units to use
    model1.add(Dense(unitsfc2))
  #Add a dropout layer
    model1.add(Dropout(drop))


  # Classifier
    model1.add(Dense(4, activation = 'softmax'))

  # Compiling the model
    model1.compile(loss = loss_function, optimizer=Adam(learning_rate=learning_rate) , m
```

```python
  # model1.summary()
  return model1

build_model(keras_tuner.HyperParameters())
```

Out[33]:
```
<keras.src.engine.sequential.Sequential at 0x7bb8c08f6f80>
```

In [34]:
```python
# Set the path to the folder where you want to save the tuner output
tuner_path = "/content/drive/MyDrive/Tuners/"

tuner = keras_tuner.RandomSearch(
    hypermodel=build_model,
    objective="val_accuracy",
    max_trials=3,
    executions_per_trial=2,
    overwrite=True,
    directory=tuner_path,
    project_name="Facial_Emotion_Milestone",
)
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 8
activation (Choice)
{'default': 'leaky-relu', 'conditions': [], 'values': ['leaky-relu', 'ReLU', 'selu'],
'ordered': False}
learning_rate (Float)
{'default': 0.0001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.01, 'step':
None, 'sampling': 'log'}
units1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units3 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units4 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
```

In [35]:
```python
tuner.search(train_set, epochs=4, validation_data=validation_set)
```

```
Trial 3 Complete [00h 02m 19s]
val_accuracy: 0.36733052134513855

Best val_accuracy So Far: 0.36733052134513855
Total elapsed time: 00h 09m 57s
```

In [36]:
```python
# Get the top model.
model1 = tuner.get_best_models(num_models=1)
best_model = model1[0]
```

```
best_model.build(input_shape=inputshape)
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 128)       1280      
                                                                 
 conv2d_1 (Conv2D)           (None, 48, 48, 32)        65568     
                                                                 
 max_pooling2d (MaxPooling2  (None, 24, 24, 32)        0         
 D)                                                              
                                                                 
 dropout (Dropout)           (None, 24, 24, 32)        0         
                                                                 
 batch_normalization (Batch  (None, 24, 24, 32)        128       
 Normalization)                                                  
                                                                 
 conv2d_2 (Conv2D)           (None, 24, 24, 64)        8256      
                                                                 
 conv2d_3 (Conv2D)           (None, 24, 24, 480)       276960    
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 480)       0         
 g2D)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 12, 12, 480)       0         
                                                                 
 batch_normalization_1 (Bat  (None, 12, 12, 480)       1920      
 chNormalization)                                                
                                                                 
 flatten (Flatten)           (None, 69120)             0         
                                                                 
 dense (Dense)               (None, 192)               13271232  
                                                                 
 dropout_2 (Dropout)         (None, 192)               0         
                                                                 
 dense_1 (Dense)             (None, 256)               49408     
                                                                 
 dropout_3 (Dropout)         (None, 256)               0         
                                                                 
 dense_2 (Dense)             (None, 4)                 1028      
                                                                 
=================================================================
Total params: 13675780 (52.17 MB)
Trainable params: 13674756 (52.17 MB)
Non-trainable params: 1024 (4.00 KB)
_________________________________________________________________
```

In [37]:

```
tuner.results_summary()
```

```
Results summary
Results in /content/drive/MyDrive/Tuners/Facial_Emotion_Milestone
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 2 summary
Hyperparameters:
activation: leaky-relu
learning_rate: 0.00010628983155127783
units1: 128
units2: 32
units3: 64
units4: 480
unitsfc1: 192
unitsfc2: 256
Score: 0.36733052134513855

Trial 0 summary
Hyperparameters:
activation: leaky-relu
learning_rate: 0.002317701860832247
units1: 512
units2: 416
units3: 416
units4: 64
unitsfc1: 352
unitsfc2: 128
Score: 0.3284047544002533

Trial 1 summary
Hyperparameters:
activation: ReLU
learning_rate: 0.0006757262292232271
units1: 320
units2: 128
units3: 256
units4: 224
unitsfc1: 224
unitsfc2: 64
Score: 0.26332730054855347
```

In [38]:

```
# Clearing backend
from tensorflow.keras import backend
backend.clear_session()

#set random seed
np.random.seed(5)
random.seed(5)
tf.random.set_seed(5)
```

In [39]:

```
#Actually training the model using the selected Tuned Parameters from above
drop=0.25

#setting parameters
model1 = Sequential()

#Trial 2 summary
#Copy and Paste Hyperparameters from Tuning output window:
activation1= 'leaky-relu'
```

```
learning_rate= 0.00010628983155127783
units1= 128
units2= 32
units3= 64
units4= 480
unitsfc1= 192
unitsfc2= 256
#Score: 0.3683946281671524
optimizer = Adam(learning_rate =learning_rate) #tuner results for learning_rate


# First Convolutional layer with 64 filters and the kernel size of 3x3. Use the 'same'
model1.add(Conv2D(filters=units1, kernel_size=(3, 3), input_shape = (inputshape), padd

# Second Convolutional Block
model1.add(Conv2D(filters=units2, kernel_size=(4, 4), padding = 'same', activation=act
#Max Pooling
model1.add(MaxPooling2D(2, 2))
#Add a dropout layer
model1.add(Dropout(drop))
#Add a BatchNormalization layer
model1.add(BatchNormalization())

# Third Convolutional Block
model1.add(Conv2D(filters=units3, kernel_size=(2, 2), padding = 'same', activation=act


# Fourth Convolutional Block
model1.add(Conv2D(filters=units4, kernel_size=(3, 3), padding = 'same', activation=act
#Max Pooling
model1.add(MaxPooling2D(2, 2))
#Add a dropout layer
model1.add(Dropout(drop))
#Add a BatchNormalization layer
model1.add(BatchNormalization())

#flatten before adding first fully connected block
model1.add(Flatten())

# First fully Connected Block
#default activation is relu
model1.add(Dense(unitsfc1))
#Add a dropout layer
model1.add(Dropout(drop))

#model1.add(Flatten())

# Second fully Connected Block
model1.add(Dense(unitsfc2))
#Add a dropout layer
model1.add(Dropout(drop))


# Classifier
model1.add(Dense(4, activation = 'softmax'))
loss_function = 'categorical_crossentropy'

model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 128)       1280

 conv2d_1 (Conv2D)           (None, 48, 48, 32)        65568

 max_pooling2d (MaxPooling2  (None, 24, 24, 32)        0
 D)

 dropout (Dropout)           (None, 24, 24, 32)        0

 batch_normalization (Batch  (None, 24, 24, 32)        128
 Normalization)

 conv2d_2 (Conv2D)           (None, 24, 24, 64)        8256

 conv2d_3 (Conv2D)           (None, 24, 24, 480)       276960

 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 480)       0
 g2D)

 dropout_1 (Dropout)         (None, 12, 12, 480)       0

 batch_normalization_1 (Bat  (None, 12, 12, 480)       1920
 chNormalization)

 flatten (Flatten)           (None, 69120)             0

 dense (Dense)               (None, 192)               13271232

 dropout_2 (Dropout)         (None, 192)               0

 dense_1 (Dense)             (None, 256)               49408

 dropout_3 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 4)                 1028

=================================================================
Total params: 13675780 (52.17 MB)
Trainable params: 13674756 (52.17 MB)
Non-trainable params: 1024 (4.00 KB)
_________________________________________________________________
```

Compiling and Training the Model

In [40]:

```
# Compiling the model
model1.compile(loss = loss_function, optimizer = optimizer , metrics = ['accuracy'])
```

In [41]:

```
# Import callbacks and define the early stopping callback
from keras.callbacks import EarlyStopping


# Set the path to the folder where you want to save the checkpoint
checkpoint_path = "/content/drive/MyDrive/Colab_Checkpoints/checkpoint.ckpt"
```

```
# Save the model's weights and optimizer state
model1.save_weights(checkpoint_path)



early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.0005, patience=7, mode=

checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                    monitor="val_accuracy", mode="max",
                    save_best_only=True, verbose=1)
```

In [42]:
```
# Fitting the model
history1 = model1.fit(
    train_set,
    validation_data = validation_set,
    callbacks=[early_stopping, checkpoint],
    epochs = 100)
    #epochs = 75)
```

```
Epoch 1/100
470/471 [============================>.] - ETA: 0s - loss: 2.6319 - accuracy: 0.2704
Epoch 1: val_accuracy improved from -inf to 0.36693, saving model to /content/drive/M
yDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 23s 43ms/step - loss: 2.6303 - accuracy:
0.2704 - val_loss: 1.4970 - val_accuracy: 0.3669
Epoch 2/100
471/471 [==============================] - ETA: 0s - loss: 1.9040 - accuracy: 0.2710
Epoch 2: val_accuracy improved from 0.36693 to 0.36753, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 37ms/step - loss: 1.9040 - accuracy:
0.2710 - val_loss: 1.6141 - val_accuracy: 0.3675
Epoch 3/100
471/471 [==============================] - ETA: 0s - loss: 1.8614 - accuracy: 0.2652
Epoch 3: val_accuracy did not improve from 0.36753
471/471 [==============================] - 16s 35ms/step - loss: 1.8614 - accuracy:
0.2652 - val_loss: 1.3876 - val_accuracy: 0.2828
Epoch 4/100
470/471 [============================>.] - ETA: 0s - loss: 1.8360 - accuracy: 0.2713
Epoch 4: val_accuracy improved from 0.36753 to 0.37035, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.8363 - accuracy:
0.2713 - val_loss: 1.7040 - val_accuracy: 0.3703
Epoch 5/100
471/471 [==============================] - ETA: 0s - loss: 1.8045 - accuracy: 0.2628
Epoch 5: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 33ms/step - loss: 1.8045 - accuracy:
0.2628 - val_loss: 1.5879 - val_accuracy: 0.3691
Epoch 6/100
470/471 [============================>.] - ETA: 0s - loss: 1.7770 - accuracy: 0.2654
Epoch 6: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 33ms/step - loss: 1.7776 - accuracy:
0.2652 - val_loss: 1.6400 - val_accuracy: 0.3681
Epoch 7/100
469/471 [============================>.] - ETA: 0s - loss: 1.7187 - accuracy: 0.2676
Epoch 7: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 33ms/step - loss: 1.7187 - accuracy:
0.2675 - val_loss: 1.5105 - val_accuracy: 0.3679
Epoch 8/100
471/471 [==============================] - ETA: 0s - loss: 1.6899 - accuracy: 0.2653
Epoch 8: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 34ms/step - loss: 1.6899 - accuracy:
0.2653 - val_loss: 1.6342 - val_accuracy: 0.2370
Epoch 9/100
470/471 [============================>.] - ETA: 0s - loss: 1.6648 - accuracy: 0.2696
Epoch 9: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 34ms/step - loss: 1.6645 - accuracy:
0.2695 - val_loss: 1.3785 - val_accuracy: 0.3681
Epoch 10/100
471/471 [==============================] - ETA: 0s - loss: 1.6246 - accuracy: 0.2727
Epoch 10: val_accuracy did not improve from 0.37035
471/471 [==============================] - 17s 35ms/step - loss: 1.6246 - accuracy:
0.2727 - val_loss: 1.6998 - val_accuracy: 0.2788
Epoch 11/100
469/471 [============================>.] - ETA: 0s - loss: 1.5988 - accuracy: 0.2650
Epoch 11: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 33ms/step - loss: 1.5979 - accuracy:
0.2653 - val_loss: 1.4773 - val_accuracy: 0.3657
Epoch 12/100
471/471 [==============================] - ETA: 0s - loss: 1.5635 - accuracy: 0.2726
```

```
Epoch 12: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 33ms/step - loss: 1.5635 - accuracy:
0.2726 - val_loss: 1.3681 - val_accuracy: 0.3637
Epoch 13/100
471/471 [==============================] - ETA: 0s - loss: 1.5389 - accuracy: 0.2739
Epoch 13: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 33ms/step - loss: 1.5389 - accuracy:
0.2739 - val_loss: 1.4375 - val_accuracy: 0.3685
Epoch 14/100
470/471 [============================>.] - ETA: 0s - loss: 1.5137 - accuracy: 0.2760
Epoch 14: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 34ms/step - loss: 1.5133 - accuracy:
0.2764 - val_loss: 1.7253 - val_accuracy: 0.3693
Epoch 15/100
470/471 [============================>.] - ETA: 0s - loss: 1.4905 - accuracy: 0.2818
Epoch 15: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 35ms/step - loss: 1.4906 - accuracy:
0.2818 - val_loss: 1.4174 - val_accuracy: 0.3671
Epoch 16/100
471/471 [==============================] - ETA: 0s - loss: 1.4824 - accuracy: 0.2697
Epoch 16: val_accuracy did not improve from 0.37035
471/471 [==============================] - 16s 34ms/step - loss: 1.4824 - accuracy:
0.2697 - val_loss: 1.4477 - val_accuracy: 0.3671
Epoch 17/100
471/471 [==============================] - ETA: 0s - loss: 1.4630 - accuracy: 0.2791
Epoch 17: val_accuracy improved from 0.37035 to 0.37095, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.4630 - accuracy:
0.2791 - val_loss: 1.3420 - val_accuracy: 0.3710
Epoch 18/100
469/471 [============================>.] - ETA: 0s - loss: 1.4454 - accuracy: 0.2860
Epoch 18: val_accuracy did not improve from 0.37095
471/471 [==============================] - 16s 34ms/step - loss: 1.4453 - accuracy:
0.2863 - val_loss: 1.5351 - val_accuracy: 0.2096
Epoch 19/100
471/471 [==============================] - ETA: 0s - loss: 1.4345 - accuracy: 0.2820
Epoch 19: val_accuracy did not improve from 0.37095
471/471 [==============================] - 16s 33ms/step - loss: 1.4345 - accuracy:
0.2820 - val_loss: 1.3365 - val_accuracy: 0.3673
Epoch 20/100
469/471 [============================>.] - ETA: 0s - loss: 1.4139 - accuracy: 0.3018
Epoch 20: val_accuracy did not improve from 0.37095
471/471 [==============================] - 16s 33ms/step - loss: 1.4140 - accuracy:
0.3020 - val_loss: 1.4586 - val_accuracy: 0.2641
Epoch 21/100
470/471 [============================>.] - ETA: 0s - loss: 1.4013 - accuracy: 0.2978
Epoch 21: val_accuracy did not improve from 0.37095
471/471 [==============================] - 16s 33ms/step - loss: 1.4013 - accuracy:
0.2980 - val_loss: 1.5605 - val_accuracy: 0.2496
Epoch 22/100
470/471 [============================>.] - ETA: 0s - loss: 1.3745 - accuracy: 0.3250
Epoch 22: val_accuracy did not improve from 0.37095
471/471 [==============================] - 16s 34ms/step - loss: 1.3745 - accuracy:
0.3249 - val_loss: 1.5553 - val_accuracy: 0.2128
Epoch 23/100
471/471 [==============================] - ETA: 0s - loss: 1.3641 - accuracy: 0.3276
Epoch 23: val_accuracy improved from 0.37095 to 0.37135, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 17s 37ms/step - loss: 1.3641 - accuracy:
0.3276 - val_loss: 1.2869 - val_accuracy: 0.3714
```

```
Epoch 24/100
470/471 [============================>.] - ETA: 0s - loss: 1.3403 - accuracy: 0.3452
Epoch 24: val_accuracy improved from 0.37135 to 0.40455, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.3405 - accuracy:
0.3453 - val_loss: 1.2715 - val_accuracy: 0.4045
Epoch 25/100
471/471 [==============================] - ETA: 0s - loss: 1.3246 - accuracy: 0.3587
Epoch 25: val_accuracy did not improve from 0.40455
471/471 [==============================] - 16s 34ms/step - loss: 1.3246 - accuracy:
0.3587 - val_loss: 1.3956 - val_accuracy: 0.3653
Epoch 26/100
470/471 [============================>.] - ETA: 0s - loss: 1.3035 - accuracy: 0.3681
Epoch 26: val_accuracy improved from 0.40455 to 0.43351, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 17s 37ms/step - loss: 1.3033 - accuracy:
0.3684 - val_loss: 1.2250 - val_accuracy: 0.4335
Epoch 27/100
471/471 [==============================] - ETA: 0s - loss: 1.2788 - accuracy: 0.3961
Epoch 27: val_accuracy improved from 0.43351 to 0.43553, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.2788 - accuracy:
0.3961 - val_loss: 1.2176 - val_accuracy: 0.4355
Epoch 28/100
471/471 [==============================] - ETA: 0s - loss: 1.2593 - accuracy: 0.4079
Epoch 28: val_accuracy improved from 0.43553 to 0.43955, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.2593 - accuracy:
0.4079 - val_loss: 1.2070 - val_accuracy: 0.4395
Epoch 29/100
471/471 [==============================] - ETA: 0s - loss: 1.2426 - accuracy: 0.4106
Epoch 29: val_accuracy improved from 0.43955 to 0.45645, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.2426 - accuracy:
0.4106 - val_loss: 1.1826 - val_accuracy: 0.4564
Epoch 30/100
471/471 [==============================] - ETA: 0s - loss: 1.2297 - accuracy: 0.4276
Epoch 30: val_accuracy improved from 0.45645 to 0.46812, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.2297 - accuracy:
0.4276 - val_loss: 1.1685 - val_accuracy: 0.4681
Epoch 31/100
469/471 [============================>.] - ETA: 0s - loss: 1.2123 - accuracy: 0.4343
Epoch 31: val_accuracy did not improve from 0.46812
471/471 [==============================] - 16s 34ms/step - loss: 1.2119 - accuracy:
0.4344 - val_loss: 1.2778 - val_accuracy: 0.4041
Epoch 32/100
470/471 [============================>.] - ETA: 0s - loss: 1.1859 - accuracy: 0.4600
Epoch 32: val_accuracy improved from 0.46812 to 0.47596, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.1862 - accuracy:
0.4597 - val_loss: 1.1721 - val_accuracy: 0.4760
Epoch 33/100
470/471 [============================>.] - ETA: 0s - loss: 1.1800 - accuracy: 0.4606
Epoch 33: val_accuracy improved from 0.47596 to 0.47636, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.1799 - accuracy:
0.4605 - val_loss: 1.1352 - val_accuracy: 0.4764
Epoch 34/100
471/471 [==============================] - ETA: 0s - loss: 1.1604 - accuracy: 0.4668
```

```
Epoch 34: val_accuracy improved from 0.47636 to 0.49950, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.1604 - accuracy:
0.4668 - val_loss: 1.1115 - val_accuracy: 0.4995
Epoch 35/100
470/471 [============================>.] - ETA: 0s - loss: 1.1452 - accuracy: 0.4803
Epoch 35: val_accuracy improved from 0.49950 to 0.50372, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.1451 - accuracy:
0.4803 - val_loss: 1.1094 - val_accuracy: 0.5037
Epoch 36/100
471/471 [==============================] - ETA: 0s - loss: 1.1327 - accuracy: 0.4873
Epoch 36: val_accuracy did not improve from 0.50372
471/471 [==============================] - 16s 34ms/step - loss: 1.1327 - accuracy:
0.4873 - val_loss: 1.1081 - val_accuracy: 0.5035
Epoch 37/100
471/471 [==============================] - ETA: 0s - loss: 1.1257 - accuracy: 0.4899
Epoch 37: val_accuracy did not improve from 0.50372
471/471 [==============================] - 16s 33ms/step - loss: 1.1257 - accuracy:
0.4899 - val_loss: 1.0996 - val_accuracy: 0.4989
Epoch 38/100
470/471 [============================>.] - ETA: 0s - loss: 1.1113 - accuracy: 0.5084
Epoch 38: val_accuracy did not improve from 0.50372
471/471 [==============================] - 16s 33ms/step - loss: 1.1116 - accuracy:
0.5084 - val_loss: 1.1099 - val_accuracy: 0.4902
Epoch 39/100
471/471 [==============================] - ETA: 0s - loss: 1.1074 - accuracy: 0.5043
Epoch 39: val_accuracy improved from 0.50372 to 0.52887, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.1074 - accuracy:
0.5043 - val_loss: 1.0456 - val_accuracy: 0.5289
Epoch 40/100
470/471 [============================>.] - ETA: 0s - loss: 1.0820 - accuracy: 0.5209
Epoch 40: val_accuracy improved from 0.52887 to 0.53309, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 37ms/step - loss: 1.0816 - accuracy:
0.5212 - val_loss: 1.0732 - val_accuracy: 0.5331
Epoch 41/100
469/471 [============================>.] - ETA: 0s - loss: 1.0713 - accuracy: 0.5266
Epoch 41: val_accuracy improved from 0.53309 to 0.56286, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.0710 - accuracy:
0.5266 - val_loss: 1.0168 - val_accuracy: 0.5629
Epoch 42/100
470/471 [============================>.] - ETA: 0s - loss: 1.0624 - accuracy: 0.5342
Epoch 42: val_accuracy did not improve from 0.56286
471/471 [==============================] - 16s 34ms/step - loss: 1.0626 - accuracy:
0.5343 - val_loss: 1.0202 - val_accuracy: 0.5472
Epoch 43/100
470/471 [============================>.] - ETA: 0s - loss: 1.0573 - accuracy: 0.5394
Epoch 43: val_accuracy improved from 0.56286 to 0.56890, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.0572 - accuracy:
0.5395 - val_loss: 0.9975 - val_accuracy: 0.5689
Epoch 44/100
471/471 [==============================] - ETA: 0s - loss: 1.0488 - accuracy: 0.5398
Epoch 44: val_accuracy did not improve from 0.56890
471/471 [==============================] - 16s 34ms/step - loss: 1.0488 - accuracy:
0.5398 - val_loss: 1.0114 - val_accuracy: 0.5598
Epoch 45/100
```

```
470/471 [============================>.] - ETA: 0s - loss: 1.0435 - accuracy: 0.5452
Epoch 45: val_accuracy improved from 0.56890 to 0.58882, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.0437 - accuracy:
0.5450 - val_loss: 0.9768 - val_accuracy: 0.5888
Epoch 46/100
471/471 [==============================] - ETA: 0s - loss: 1.0310 - accuracy: 0.5522
Epoch 46: val_accuracy did not improve from 0.58882
471/471 [==============================] - 16s 34ms/step - loss: 1.0310 - accuracy:
0.5522 - val_loss: 0.9806 - val_accuracy: 0.5796
Epoch 47/100
469/471 [============================>.] - ETA: 0s - loss: 1.0260 - accuracy: 0.5561
Epoch 47: val_accuracy improved from 0.58882 to 0.59042, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.0253 - accuracy:
0.5567 - val_loss: 0.9658 - val_accuracy: 0.5904
Epoch 48/100
470/471 [============================>.] - ETA: 0s - loss: 1.0250 - accuracy: 0.5575
Epoch 48: val_accuracy improved from 0.59042 to 0.59344, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.0249 - accuracy:
0.5576 - val_loss: 0.9603 - val_accuracy: 0.5934
Epoch 49/100
470/471 [============================>.] - ETA: 0s - loss: 1.0118 - accuracy: 0.5641
Epoch 49: val_accuracy improved from 0.59344 to 0.59767, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.0116 - accuracy:
0.5640 - val_loss: 0.9378 - val_accuracy: 0.5977
Epoch 50/100
470/471 [============================>.] - ETA: 0s - loss: 1.0099 - accuracy: 0.5664
Epoch 50: val_accuracy did not improve from 0.59767
471/471 [==============================] - 17s 36ms/step - loss: 1.0096 - accuracy:
0.5666 - val_loss: 0.9683 - val_accuracy: 0.5812
Epoch 51/100
471/471 [==============================] - ETA: 0s - loss: 1.0013 - accuracy: 0.5704
Epoch 51: val_accuracy improved from 0.59767 to 0.60591, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 37ms/step - loss: 1.0013 - accuracy:
0.5704 - val_loss: 0.9395 - val_accuracy: 0.6059
Epoch 52/100
471/471 [==============================] - ETA: 0s - loss: 1.0001 - accuracy: 0.5692
Epoch 52: val_accuracy did not improve from 0.60591
471/471 [==============================] - 16s 34ms/step - loss: 1.0001 - accuracy:
0.5692 - val_loss: 0.9543 - val_accuracy: 0.5928
Epoch 53/100
470/471 [============================>.] - ETA: 0s - loss: 0.9931 - accuracy: 0.5766
Epoch 53: val_accuracy improved from 0.60591 to 0.61637, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.9936 - accuracy:
0.5765 - val_loss: 0.9259 - val_accuracy: 0.6164
Epoch 54/100
471/471 [==============================] - ETA: 0s - loss: 0.9912 - accuracy: 0.5737
Epoch 54: val_accuracy did not improve from 0.61637
471/471 [==============================] - 16s 34ms/step - loss: 0.9912 - accuracy:
0.5737 - val_loss: 0.9264 - val_accuracy: 0.6130
Epoch 55/100
471/471 [==============================] - ETA: 0s - loss: 0.9797 - accuracy: 0.5791
Epoch 55: val_accuracy did not improve from 0.61637
471/471 [==============================] - 16s 34ms/step - loss: 0.9797 - accuracy:
0.5791 - val_loss: 1.0093 - val_accuracy: 0.5627
```

```
Epoch 56/100
471/471 [==============================] - ETA: 0s - loss: 0.9807 - accuracy: 0.5838
Epoch 56: val_accuracy did not improve from 0.61637
471/471 [==============================] - 16s 34ms/step - loss: 0.9807 - accuracy:
0.5838 - val_loss: 0.9269 - val_accuracy: 0.6152
Epoch 57/100
471/471 [==============================] - ETA: 0s - loss: 0.9838 - accuracy: 0.5784
Epoch 57: val_accuracy did not improve from 0.61637
471/471 [==============================] - 16s 34ms/step - loss: 0.9838 - accuracy:
0.5784 - val_loss: 0.9463 - val_accuracy: 0.5987
Epoch 58/100
471/471 [==============================] - ETA: 0s - loss: 0.9777 - accuracy: 0.5797
Epoch 58: val_accuracy did not improve from 0.61637
471/471 [==============================] - 16s 33ms/step - loss: 0.9777 - accuracy:
0.5797 - val_loss: 0.9170 - val_accuracy: 0.6156
Epoch 59/100
469/471 [============================>.] - ETA: 0s - loss: 0.9651 - accuracy: 0.5889
Epoch 59: val_accuracy improved from 0.61637 to 0.61758, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.9649 - accuracy:
0.5888 - val_loss: 0.9106 - val_accuracy: 0.6176
Epoch 60/100
470/471 [============================>.] - ETA: 0s - loss: 0.9667 - accuracy: 0.5861
Epoch 60: val_accuracy did not improve from 0.61758
471/471 [==============================] - 16s 34ms/step - loss: 0.9668 - accuracy:
0.5860 - val_loss: 0.9299 - val_accuracy: 0.6061
Epoch 61/100
470/471 [============================>.] - ETA: 0s - loss: 0.9667 - accuracy: 0.5915
Epoch 61: val_accuracy did not improve from 0.61758
471/471 [==============================] - 16s 33ms/step - loss: 0.9667 - accuracy:
0.5915 - val_loss: 0.9166 - val_accuracy: 0.6150
Epoch 62/100
469/471 [============================>.] - ETA: 0s - loss: 0.9601 - accuracy: 0.5917
Epoch 62: val_accuracy did not improve from 0.61758
471/471 [==============================] - 16s 33ms/step - loss: 0.9609 - accuracy:
0.5915 - val_loss: 0.9411 - val_accuracy: 0.6021
Epoch 63/100
471/471 [==============================] - ETA: 0s - loss: 0.9633 - accuracy: 0.5898
Epoch 63: val_accuracy did not improve from 0.61758
471/471 [==============================] - 16s 33ms/step - loss: 0.9633 - accuracy:
0.5898 - val_loss: 0.9202 - val_accuracy: 0.6132
Epoch 64/100
471/471 [==============================] - ETA: 0s - loss: 0.9557 - accuracy: 0.5917
Epoch 64: val_accuracy did not improve from 0.61758
471/471 [==============================] - 16s 34ms/step - loss: 0.9557 - accuracy:
0.5917 - val_loss: 0.9234 - val_accuracy: 0.6166
Epoch 65/100
471/471 [==============================] - ETA: 0s - loss: 0.9594 - accuracy: 0.5916
Epoch 65: val_accuracy improved from 0.61758 to 0.63126, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 37ms/step - loss: 0.9594 - accuracy:
0.5916 - val_loss: 0.8821 - val_accuracy: 0.6313
Epoch 66/100
471/471 [==============================] - ETA: 0s - loss: 0.9518 - accuracy: 0.5919
Epoch 66: val_accuracy did not improve from 0.63126
471/471 [==============================] - 17s 37ms/step - loss: 0.9518 - accuracy:
0.5919 - val_loss: 0.9034 - val_accuracy: 0.6232
Epoch 67/100
469/471 [============================>.] - ETA: 0s - loss: 0.9475 - accuracy: 0.6000
Epoch 67: val_accuracy improved from 0.63126 to 0.63186, saving model to /content/dri
```

```
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.9475 - accuracy:
0.6000 - val_loss: 0.8986 - val_accuracy: 0.6319
Epoch 68/100
470/471 [============================>.] - ETA: 0s - loss: 0.9455 - accuracy: 0.6028
Epoch 68: val_accuracy did not improve from 0.63186
471/471 [==============================] - 18s 37ms/step - loss: 0.9460 - accuracy:
0.6027 - val_loss: 0.8976 - val_accuracy: 0.6214
Epoch 69/100
471/471 [==============================] - ETA: 0s - loss: 0.9405 - accuracy: 0.6023
Epoch 69: val_accuracy did not improve from 0.63186
471/471 [==============================] - 16s 35ms/step - loss: 0.9405 - accuracy:
0.6023 - val_loss: 0.8924 - val_accuracy: 0.6284
Epoch 70/100
471/471 [==============================] - ETA: 0s - loss: 0.9441 - accuracy: 0.6048
Epoch 70: val_accuracy improved from 0.63186 to 0.63770, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 37ms/step - loss: 0.9441 - accuracy:
0.6048 - val_loss: 0.8690 - val_accuracy: 0.6377
Epoch 71/100
471/471 [==============================] - ETA: 0s - loss: 0.9338 - accuracy: 0.6065
Epoch 71: val_accuracy did not improve from 0.63770
471/471 [==============================] - 16s 35ms/step - loss: 0.9338 - accuracy:
0.6065 - val_loss: 0.8817 - val_accuracy: 0.6365
Epoch 72/100
469/471 [============================>.] - ETA: 0s - loss: 0.9295 - accuracy: 0.6035
Epoch 72: val_accuracy did not improve from 0.63770
471/471 [==============================] - 16s 33ms/step - loss: 0.9296 - accuracy:
0.6034 - val_loss: 0.9407 - val_accuracy: 0.6192
Epoch 73/100
469/471 [============================>.] - ETA: 0s - loss: 0.9298 - accuracy: 0.6055
Epoch 73: val_accuracy improved from 0.63770 to 0.64172, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.9299 - accuracy:
0.6056 - val_loss: 0.8756 - val_accuracy: 0.6417
Epoch 74/100
471/471 [==============================] - ETA: 0s - loss: 0.9359 - accuracy: 0.6032
Epoch 74: val_accuracy improved from 0.64172 to 0.64735, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 37ms/step - loss: 0.9359 - accuracy:
0.6032 - val_loss: 0.8637 - val_accuracy: 0.6474
Epoch 75/100
470/471 [============================>.] - ETA: 0s - loss: 0.9283 - accuracy: 0.6074
Epoch 75: val_accuracy did not improve from 0.64735
471/471 [==============================] - 16s 34ms/step - loss: 0.9281 - accuracy:
0.6074 - val_loss: 0.9450 - val_accuracy: 0.6037
Epoch 76/100
471/471 [==============================] - ETA: 0s - loss: 0.9259 - accuracy: 0.6093
Epoch 76: val_accuracy did not improve from 0.64735
471/471 [==============================] - 16s 34ms/step - loss: 0.9259 - accuracy:
0.6093 - val_loss: 0.8743 - val_accuracy: 0.6321
Epoch 77/100
471/471 [==============================] - ETA: 0s - loss: 0.9294 - accuracy: 0.6058
Epoch 77: val_accuracy improved from 0.64735 to 0.65379, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 17s 37ms/step - loss: 0.9294 - accuracy:
0.6058 - val_loss: 0.8545 - val_accuracy: 0.6538
Epoch 78/100
471/471 [==============================] - ETA: 0s - loss: 0.9289 - accuracy: 0.6062
Epoch 78: val_accuracy improved from 0.65379 to 0.65399, saving model to /content/dri
```

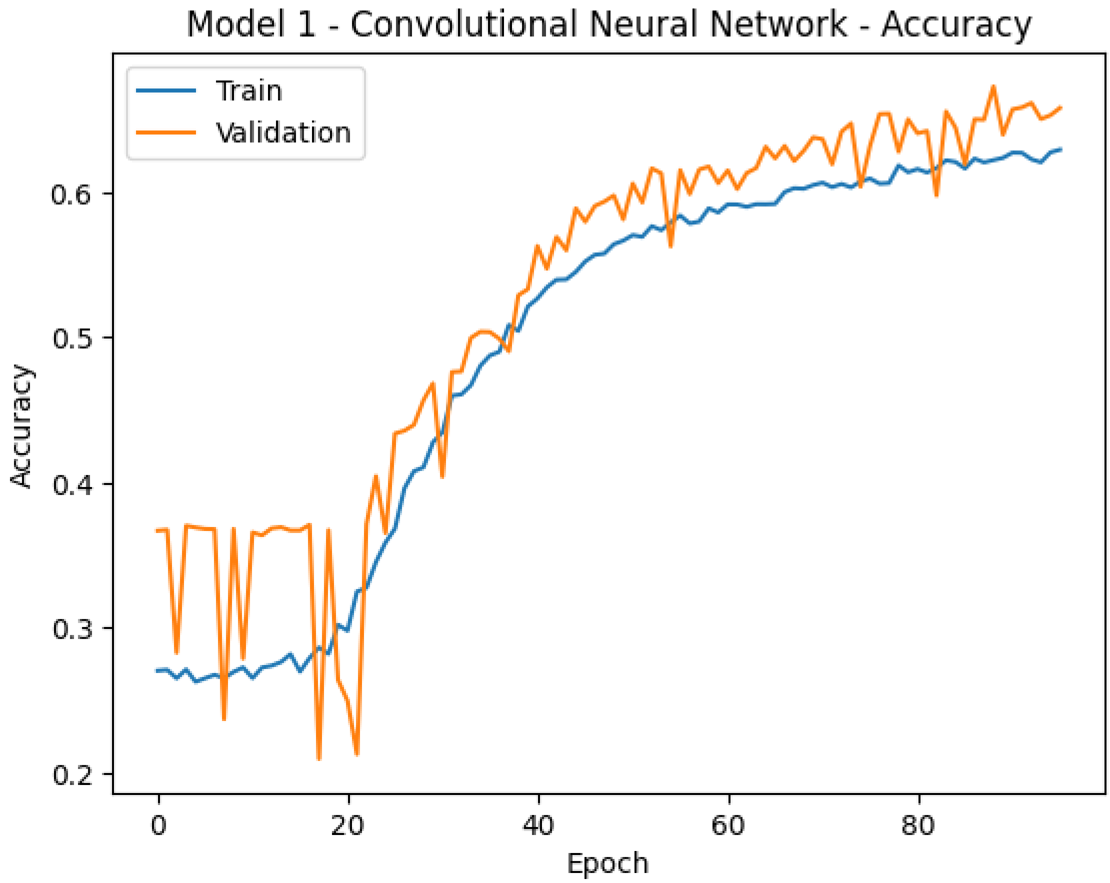
```
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.9289 - accuracy: 0.6062 - val_loss: 0.8539 - val_accuracy: 0.6540
Epoch 79/100
471/471 [==============================] - ETA: 0s - loss: 0.9185 - accuracy: 0.6184
Epoch 79: val_accuracy did not improve from 0.65399
471/471 [==============================] - 16s 34ms/step - loss: 0.9185 - accuracy: 0.6184 - val_loss: 0.9083 - val_accuracy: 0.6278
Epoch 80/100
470/471 [============================>.] - ETA: 0s - loss: 0.9164 - accuracy: 0.6139
Epoch 80: val_accuracy did not improve from 0.65399
471/471 [==============================] - 16s 34ms/step - loss: 0.9162 - accuracy: 0.6137 - val_loss: 0.8564 - val_accuracy: 0.6502
Epoch 81/100
469/471 [============================>.] - ETA: 0s - loss: 0.9112 - accuracy: 0.6163
Epoch 81: val_accuracy did not improve from 0.65399
471/471 [==============================] - 16s 33ms/step - loss: 0.9122 - accuracy: 0.6159 - val_loss: 0.8628 - val_accuracy: 0.6405
Epoch 82/100
471/471 [==============================] - ETA: 0s - loss: 0.9168 - accuracy: 0.6135
Epoch 82: val_accuracy did not improve from 0.65399
471/471 [==============================] - 16s 34ms/step - loss: 0.9168 - accuracy: 0.6135 - val_loss: 0.8506 - val_accuracy: 0.6421
Epoch 83/100
471/471 [==============================] - ETA: 0s - loss: 0.9127 - accuracy: 0.6166
Epoch 83: val_accuracy did not improve from 0.65399
471/471 [==============================] - 16s 33ms/step - loss: 0.9127 - accuracy: 0.6166 - val_loss: 0.9618 - val_accuracy: 0.5977
Epoch 84/100
471/471 [==============================] - ETA: 0s - loss: 0.9094 - accuracy: 0.6220
Epoch 84: val_accuracy improved from 0.65399 to 0.65560, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.9094 - accuracy: 0.6220 - val_loss: 0.8434 - val_accuracy: 0.6556
Epoch 85/100
471/471 [==============================] - ETA: 0s - loss: 0.9049 - accuracy: 0.6208
Epoch 85: val_accuracy did not improve from 0.65560
471/471 [==============================] - 16s 33ms/step - loss: 0.9049 - accuracy: 0.6208 - val_loss: 0.8614 - val_accuracy: 0.6443
Epoch 86/100
471/471 [==============================] - ETA: 0s - loss: 0.9071 - accuracy: 0.6162
Epoch 86: val_accuracy did not improve from 0.65560
471/471 [==============================] - 16s 34ms/step - loss: 0.9071 - accuracy: 0.6162 - val_loss: 0.9160 - val_accuracy: 0.6192
Epoch 87/100
469/471 [============================>.] - ETA: 0s - loss: 0.9105 - accuracy: 0.6229
Epoch 87: val_accuracy did not improve from 0.65560
471/471 [==============================] - 16s 33ms/step - loss: 0.9104 - accuracy: 0.6231 - val_loss: 0.8500 - val_accuracy: 0.6502
Epoch 88/100
471/471 [==============================] - ETA: 0s - loss: 0.9028 - accuracy: 0.6205
Epoch 88: val_accuracy did not improve from 0.65560
471/471 [==============================] - 16s 34ms/step - loss: 0.9028 - accuracy: 0.6205 - val_loss: 0.8367 - val_accuracy: 0.6500
Epoch 89/100
470/471 [============================>.] - ETA: 0s - loss: 0.9027 - accuracy: 0.6219
Epoch 89: val_accuracy improved from 0.65560 to 0.67290, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint.ckpt
471/471 [==============================] - 18s 37ms/step - loss: 0.9023 - accuracy: 0.6221 - val_loss: 0.8087 - val_accuracy: 0.6729
```

```
Epoch 90/100
470/471 [============================>.] - ETA: 0s - loss: 0.8998 - accuracy: 0.6235
Epoch 90: val_accuracy did not improve from 0.67290
471/471 [==============================] - 16s 34ms/step - loss: 0.8996 - accuracy:
0.6236 - val_loss: 0.8666 - val_accuracy: 0.6395
Epoch 91/100
469/471 [============================>.] - ETA: 0s - loss: 0.8981 - accuracy: 0.6273
Epoch 91: val_accuracy did not improve from 0.67290
471/471 [==============================] - 16s 33ms/step - loss: 0.8980 - accuracy:
0.6272 - val_loss: 0.8350 - val_accuracy: 0.6570
Epoch 92/100
470/471 [============================>.] - ETA: 0s - loss: 0.8980 - accuracy: 0.6271
Epoch 92: val_accuracy did not improve from 0.67290
471/471 [==============================] - 16s 34ms/step - loss: 0.8979 - accuracy:
0.6271 - val_loss: 0.8246 - val_accuracy: 0.6584
Epoch 93/100
471/471 [==============================] - ETA: 0s - loss: 0.9038 - accuracy: 0.6226
Epoch 93: val_accuracy did not improve from 0.67290
471/471 [==============================] - 16s 33ms/step - loss: 0.9038 - accuracy:
0.6226 - val_loss: 0.8377 - val_accuracy: 0.6614
Epoch 94/100
470/471 [============================>.] - ETA: 0s - loss: 0.8979 - accuracy: 0.6206
Epoch 94: val_accuracy did not improve from 0.67290
471/471 [==============================] - 16s 34ms/step - loss: 0.8982 - accuracy:
0.6206 - val_loss: 0.8415 - val_accuracy: 0.6504
Epoch 95/100
469/471 [============================>.] - ETA: 0s - loss: 0.8964 - accuracy: 0.6273
Epoch 95: val_accuracy did not improve from 0.67290
471/471 [==============================] - 16s 33ms/step - loss: 0.8964 - accuracy:
0.6274 - val_loss: 0.8399 - val_accuracy: 0.6530
Epoch 96/100
470/471 [============================>.] - ETA: 0s - loss: 0.8881 - accuracy: 0.6295
Epoch 96: val_accuracy did not improve from 0.67290
471/471 [==============================] - 16s 34ms/step - loss: 0.8881 - accuracy:
0.6293 - val_loss: 0.8290 - val_accuracy: 0.6580
```

Evaluating the Model on the Test Set

In [43]:

```python
#plot the training and validation accuracy using history log
plt.plot(history1.history['accuracy'])

plt.plot(history1.history['val_accuracy'])

plt.title('Model 1 - Convolutional Neural Network - Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc = 'upper left')

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



In [44]:
```
#set all of the random seeds
np.random.seed(5)
random.seed(5)
tf.random.set_seed(5)

# evaluate test accuracy
accuracy = model1.evaluate(test_set, verbose = 2)
```

```
4/4 - 0s - loss: 0.7555 - accuracy: 0.6797 - 201ms/epoch - 50ms/step
```

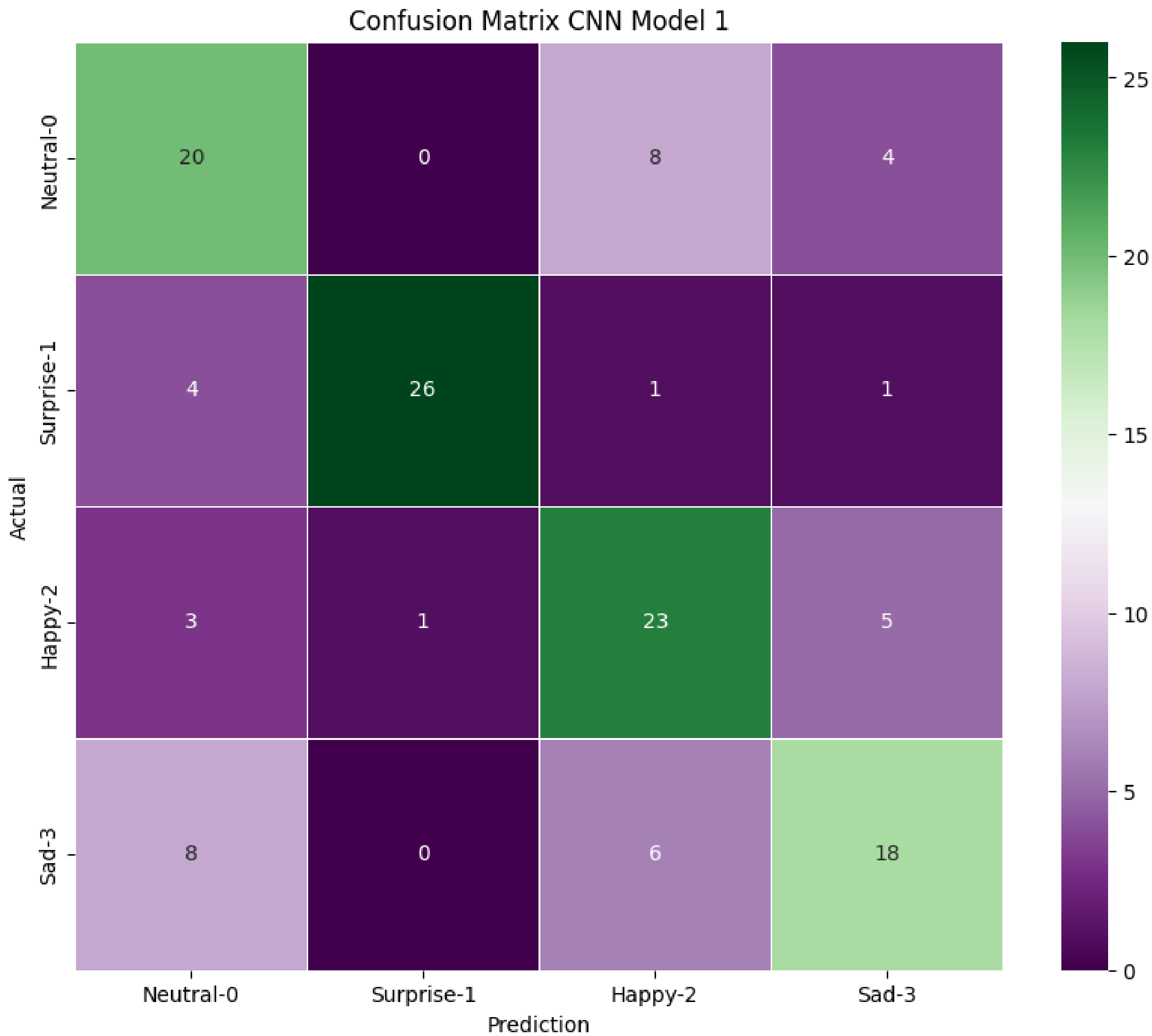
In [45]:
```
#code to get pairs of predicted vs. true for the confusion matrix
all_y_pred = []
all_y_true = []

for i in range(len(test_set)):
    x , y = test_set[i] #step through each test image/label
    y_pred = model1.predict(x) #run image though model, return predicted label

    #all_x.append(x)
    all_y_pred.append(y_pred) #add the predicted label to an ordered list
    all_y_true.append(y)      #add the true label to an ordered list

all_y_pred = np.concatenate(all_y_pred, axis=0) #concatenate all preds
all_y_true = np.concatenate(all_y_true, axis=0) #concatenate all base vals


#test_images, test_labels = next(test_set)
#pred = model1.predict(test_images)
pred = np.argmax(all_y_pred, axis=1) #get class indices
y_true = np.argmax(all_y_true, axis=1) #get class indices
```

```
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
```

In [46]:

```python
# Printing the classification report
#importing function from sklearn
from sklearn.metrics import classification_report
print(classification_report(y_true, pred))

# Plotting the Confusion Matrix using confusion matrix() function which is also predef
confusion_matrix = tf.math.confusion_matrix(y_true,pred)
f, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(
    confusion_matrix,
    annot=True,
    linewidths=.4,
    fmt="d",
    square=True,
    ax=ax,
    cmap="PRGn",
    xticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3'],
    yticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3']
)
plt.xlabel('Prediction',fontsize=10)
plt.ylabel('Actual',fontsize=10)
plt.title('Confusion Matrix CNN Model 1', fontsize=12)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.57      0.62      0.60        32
           1       0.96      0.81      0.88        32
           2       0.61      0.72      0.66        32
           3       0.64      0.56      0.60        32

    accuracy                           0.68       128
   macro avg       0.70      0.68      0.68       128
weighted avg       0.70      0.68      0.68       128
```



Observations and Insights:

The proposed model appears to learn well, but slowly. In fact, the model struggles to stabilize until it reaches ~epoch 20. At this point in the training process, training success increases dramatically (comparatively) until progress starts to taper at epoch 50.

The initial number of iterations/epochs for the model was thirty (30). The model successfully learned without signs of overfitting at 30 epochs, so the current model (model1) was revised to include early stopping callbacks.

In spite of diminishing returns in the model accuracy around epoch 50, the model was able to continue training for ~40 more epochs without any risk of overfitting. Eventually, meaningful gains in validation accuracy stopped at epoch 89; the model stopped shortly thereafter.

The model was able to achieve around 63% training and 67% validation and test accuracy without overfitting.

The reviewers of the training set (initial data provider+me) appear to have encountered a high level of ambiguity regarding the neutral dataset, which can be problematic. IT IS HIGHLY RECOMMENDED, THEREFORE, THAT GOING FORWARD, a committee of individuals (at least two or three) reviews and approves each image of the training data and recommends where each image should be placed in each class.

Given the training data and this model, the only classification with highly reliable results is "Class 1: surprised" with an F1 score of 88%. This is unsurprising, given that there are clear indicators of the *surprised* emotion - specifically, wide open mouth and eyes.

All of the images in this dataset are in grayscale mode, however, the model was tested with RGB as well as grayscale. The models performed similarly in terms of val_accuracy, however the grayscale model -unsurprisingly- outperformed RGB in terms of speed and learning rate due to the reduced complexity of information and color-space. It is unlikely that diversity in the facial hue contributes to accuracy in interpretation of facial emotion.

Creating the second Convolutional Neural Network; Model 2

- Try out a slightly larger architecture

In [47]:
```
# Clearing backend
backend.clear_session()

#set random seed
np.random.seed(5)
random.seed(5)
tf.random.set_seed(5)
```

In [48]:
```
#hyperparameter tuner
#https://keras.io/guides/keras_tuner/getting_started/

#add leaky-relu parameter alpha=0.1 to the list of activation functions that are able
#from tensorflow.keras.utils import get_custom_objects
#get_custom_objects().update({'leaky-relu': Activation(LeakyReLU(alpha=0.1))})

#define drop rate
drop=0.25

#define possible activation parameters for tuning
activation1 = "leaky-relu"
activation2 = "ReLU"
activation3 = "selu"

#define loss function for compiling the model
loss_function = 'categorical_crossentropy'

#tuning function
def build_model2(hp):

  activation=hp.Choice("activation", [activation1, activation2, activation3]) #tune th
  learning_rate = hp.Float("learning_rate", min_value=1e-4, max_value=1e-2, sampling='

# initialize model 2
  model2 = Sequential()

# First Convolutional Block
```

```
    units1=hp.Int("units1", min_value=32, max_value=512, step=32) #tune units to use for
    model2.add(Conv2D(units1, kernel_size=(3, 3), input_shape = (48, 48, 1), padding = '

  # Second Convolutional Block
    units2=hp.Int("units2", min_value=32, max_value=512, step=32) #tune units to use for
    model2.add(Conv2D(units2, kernel_size=(3, 3), padding = 'same', activation=activatic
  # Max Pooling
    model2.add(MaxPooling2D(2, 2))
  # Add a dropout layer
    model2.add(Dropout(drop))
  # Add a BatchNormalization layer
    model2.add(BatchNormalization())

  # Third Convolutional Block
    units3=hp.Int("units3", min_value=32, max_value=512, step=32) #tune units to use for
    model2.add(Conv2D(units3, kernel_size=(2, 2), padding = 'same', activation=activatic
  # Max Pooling
    model2.add(MaxPooling2D(2, 2))
  # Add a dropout layer
    model2.add(Dropout(drop))
  # Add a BatchNormalization layer
    model2.add(BatchNormalization())

  # Fourth Convolutional Block
    units4=hp.Int("units4", min_value=32, max_value=512, step=32) #tune units to use for
    model2.add(Conv2D(units4, kernel_size=(3, 3), padding = 'same', activation=activatic
  #Max Pooling
    model2.add(MaxPooling2D(2, 2))
  #Add a dropout layer
    model2.add(Dropout(drop))
  #Add a BatchNormalization layer
    model2.add(BatchNormalization())

    # Fifth Convolutional Block
    units5=hp.Int("units5", min_value=32, max_value=512, step=32) #tune units to use for
    model2.add(Conv2D(units5, kernel_size=(3, 3), padding = 'same', activation=activatic
  #Max Pooling
    model2.add(MaxPooling2D(2, 2))
  #Add a dropout layer
    model2.add(Dropout(drop))
  #Add a BatchNormalization layer
    model2.add(BatchNormalization())

  #flatten
    model2.add(Flatten())

  # First fully Connected Block
  #default activation is relu
    unitsfc1=hp.Int("unitsfc1", min_value=32, max_value=512, step=32) #tune units to use
    model2.add(Dense(unitsfc1))
  #Add a dropout layer
    model2.add(Dropout(drop))


  # Second fully Connected Block
    unitsfc2=hp.Int("unitsfc2", min_value=32, max_value=512, step=32) #tune units to use
    model2.add(Dense(unitsfc2))
  #Add a dropout layer
    model2.add(Dropout(drop))
```

```
  # Third fully Connected Block
  unitsfc3=hp.Int("unitsfc3", min_value=32, max_value=512, step=32) #tune units to use
  model2.add(Dense(unitsfc3))
#Add a dropout layer
  model2.add(Dropout(drop))


# Classifier
  model2.add(Dense(4, activation = 'softmax'))

# Compiling the model
  model2.compile(loss = loss_function, optimizer=Adam(learning_rate=learning_rate) , m

 # model1.summary()
  return model2

build_model2(keras_tuner.HyperParameters())
```

Out[48]:
```
<keras.src.engine.sequential.Sequential at 0x7bb8705a2530>
```

In [49]:
```
# Set the path to the folder where you want to save the tuner output
tuner_path2 = "/content/drive/MyDrive/Tuners/"

tuner2 = keras_tuner.RandomSearch(
    hypermodel=build_model2,
    objective="val_accuracy",
    max_trials=3,
    executions_per_trial=2,
    overwrite=True,
    directory=tuner_path2,
    project_name="Facial_Emotion_Milestone",
)
tuner2.search_space_summary()
```

```
Search space summary
Default search space size: 10
activation (Choice)
{'default': 'leaky-relu', 'conditions': [], 'values': ['leaky-relu', 'ReLU', 'selu'],
'ordered': False}
learning_rate (Float)
{'default': 0.0001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.01, 'step':
None, 'sampling': 'log'}
units1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units3 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units4 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units5 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc3 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
```

In [50]:

```
tuner2.search(train_set, epochs=4, validation_data=validation_set)
```

```
Trial 3 Complete [00h 02m 19s]
val_accuracy: 0.3630054295063019

Best val_accuracy So Far: 0.3703480064868927
Total elapsed time: 00h 07m 58s
```

In [51]:

```
# Get the top model.
model2 = tuner2.get_best_models(num_models=1)
best_model2 = model2[0]
best_model2.build(input_shape=(48,48,1))
best_model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 256)       2560      
                                                                 
 conv2d_1 (Conv2D)           (None, 48, 48, 224)       516320    
                                                                 
 max_pooling2d (MaxPooling2  (None, 24, 24, 224)       0         
 D)                                                              
                                                                 
 dropout (Dropout)           (None, 24, 24, 224)       0         
                                                                 
 batch_normalization (Batch  (None, 24, 24, 224)       896       
 Normalization)                                                  
                                                                 
 conv2d_2 (Conv2D)           (None, 24, 24, 224)       200928    
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 224)       0         
 g2D)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 12, 12, 224)       0         
                                                                 
 batch_normalization_1 (Bat  (None, 12, 12, 224)       896       
 chNormalization)                                                
                                                                 
 conv2d_3 (Conv2D)           (None, 12, 12, 64)        129088    
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 64)          0         
 g2D)                                                            
                                                                 
 dropout_2 (Dropout)         (None, 6, 6, 64)          0         
                                                                 
 batch_normalization_2 (Bat  (None, 6, 6, 64)          256       
 chNormalization)                                                
                                                                 
 conv2d_4 (Conv2D)           (None, 6, 6, 96)          55392     
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 3, 3, 96)          0         
 g2D)                                                            
                                                                 
 dropout_3 (Dropout)         (None, 3, 3, 96)          0         
                                                                 
 batch_normalization_3 (Bat  (None, 3, 3, 96)          384       
 chNormalization)                                                
                                                                 
 flatten (Flatten)           (None, 864)               0         
                                                                 
 dense (Dense)               (None, 32)                27680     
                                                                 
 dropout_4 (Dropout)         (None, 32)                0         
                                                                 
 dense_1 (Dense)             (None, 128)               4224      
                                                                 
 dropout_5 (Dropout)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 32)                4128      
                                                                 
 dropout_6 (Dropout)         (None, 32)                0         
```

```
 dense_3 (Dense)             (None, 4)                 132

=================================================================
Total params: 942884 (3.60 MB)
Trainable params: 941668 (3.59 MB)
Non-trainable params: 1216 (4.75 KB)
_________________________________________________________________
```

In [52]:
```
tuner2.results_summary()
```

```
Results summary
Results in /content/drive/MyDrive/Tuners/Facial_Emotion_Milestone
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 1 summary
Hyperparameters:
activation: ReLU
learning_rate: 0.00025717990963772296
units1: 256
units2: 224
units3: 224
units4: 64
units5: 96
unitsfc1: 32
unitsfc2: 128
unitsfc3: 32
Score: 0.3703480064868927

Trial 2 summary
Hyperparameters:
activation: leaky-relu
learning_rate: 0.006489108709865806
units1: 192
units2: 256
units3: 448
units4: 320
units5: 32
unitsfc1: 416
unitsfc2: 416
unitsfc3: 288
Score: 0.3630054295063019

Trial 0 summary
Hyperparameters:
activation: leaky-relu
learning_rate: 0.002317701860832247
units1: 512
units2: 416
units3: 416
units4: 64
units5: 352
unitsfc1: 128
unitsfc2: 256
unitsfc3: 224
Score: 0.3223697394132614
```

Paste results of best tuned parameters where indicated below:

In [53]:
```
#set the drop ratio for conv layers
drop=0.25

#initialize model
model2 = Sequential()

#setting hyperparameters
#PASTE RESULTS OF BEST TUNED PARAMETERS BELOW:
#Trial 1 summary
activation1= 'relu'
learning_rate= 0.00025717990963772296
units1= 256
units2= 224
units3= 224
units4= 64
units5= 96
unitsfc1= 32
unitsfc2= 128
unitsfc3= 32

optimizer = Adam(learning_rate =learning_rate) #tuner results for learning_rate

# First Convolutional layer with 32 filters and the kernel size of 3x3. Use the 'same'
model2.add(Conv2D(units1, kernel_size=(3, 3), input_shape = (48, 48, 1), padding = 'sa
#Max Pooling
model2.add(MaxPooling2D(2, 2))

# Second Convolutional Block
model2.add(Conv2D(units2, kernel_size=(4, 4), padding = 'same', activation=activation1
#Max Pooling
model2.add(MaxPooling2D(2, 2))
#Add a dropout layer
model2.add(Dropout(drop))
#Add a BatchNormalization layer
model2.add(BatchNormalization())

# Third Convolutional Block
model2.add(Conv2D(units3, kernel_size=(2, 2), padding = 'same', activation=activation1
#Max Pooling
model2.add(MaxPooling2D(2, 2))
#Add a dropout layer
model2.add(Dropout(drop))
#Add a BatchNormalization layer
model2.add(BatchNormalization())

# Fourth Convolutional Block
model2.add(Conv2D(units4, kernel_size=(3, 3), padding = 'same', activation=activation1
#Max Pooling
model2.add(MaxPooling2D(2, 2))
#Add a dropout layer
model2.add(Dropout(drop))
#Add a BatchNormalization layer
model2.add(BatchNormalization())

# Fifth Convolutional Block
model2.add(Conv2D(units5, kernel_size=(3, 3), padding = 'same', activation=activation1
#Max Pooling
model2.add(MaxPooling2D(2, 2))
#Add a dropout layer
```

```
model2.add(Dropout(drop))
#Add a BatchNormalization layer
model2.add(BatchNormalization())

#Flatten for dense layers
model2.add(Flatten())

# First fully Connected Block
#default activation is relu
model2.add(Dense(unitsfc1))
#Add a dropout layer
model2.add(Dropout(drop))

# Second fully Connected Block
model2.add(Dense(unitsfc2))
#Add a dropout layer
model2.add(Dropout(drop))

# Third fully Connected Block
model2.add(Dense(unitsfc3))
#Add a dropout layer
model2.add(Dropout(drop))


# Classifier Block
model2.add(Dense(4, activation = 'softmax'))
loss_function = 'categorical_crossentropy'

model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_5 (Conv2D)           (None, 48, 48, 256)       2560      
                                                                 
 max_pooling2d_4 (MaxPoolin  (None, 24, 24, 256)       0         
 g2D)                                                            
                                                                 
 conv2d_6 (Conv2D)           (None, 24, 24, 224)       917728    
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 12, 12, 224)       0         
 g2D)                                                            
                                                                 
 dropout_7 (Dropout)         (None, 12, 12, 224)       0         
                                                                 
 batch_normalization_4 (Bat  (None, 12, 12, 224)       896       
 chNormalization)                                                
                                                                 
 conv2d_7 (Conv2D)           (None, 12, 12, 224)       200928    
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 6, 6, 224)         0         
 g2D)                                                            
                                                                 
 dropout_8 (Dropout)         (None, 6, 6, 224)         0         
                                                                 
 batch_normalization_5 (Bat  (None, 6, 6, 224)         896       
 chNormalization)                                                
                                                                 
 conv2d_8 (Conv2D)           (None, 6, 6, 64)          129088    
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 3, 3, 64)          0         
 g2D)                                                            
                                                                 
 dropout_9 (Dropout)         (None, 3, 3, 64)          0         
                                                                 
 batch_normalization_6 (Bat  (None, 3, 3, 64)          256       
 chNormalization)                                                
                                                                 
 conv2d_9 (Conv2D)           (None, 3, 3, 96)          55392     
                                                                 
 max_pooling2d_8 (MaxPoolin  (None, 1, 1, 96)          0         
 g2D)                                                            
                                                                 
 dropout_10 (Dropout)        (None, 1, 1, 96)          0         
                                                                 
 batch_normalization_7 (Bat  (None, 1, 1, 96)          384       
 chNormalization)                                                
                                                                 
 flatten_1 (Flatten)         (None, 96)                0         
                                                                 
 dense_4 (Dense)             (None, 32)                3104      
                                                                 
 dropout_11 (Dropout)        (None, 32)                0         
                                                                 
 dense_5 (Dense)             (None, 128)               4224      
                                                                 
 dropout_12 (Dropout)        (None, 128)               0         
                                                                 
 dense_6 (Dense)             (None, 32)                4128      
```

```
 dropout_13 (Dropout)        (None, 32)                0

 dense_7 (Dense)             (None, 4)                 132

=================================================================
Total params: 1319716 (5.03 MB)
Trainable params: 1318500 (5.03 MB)
Non-trainable params: 1216 (4.75 KB)
_________________________________________________________________
```

Compiling and Training the Model

In [54]:
```
# Compiling the model
model2.compile(loss = loss_function, optimizer = optimizer , metrics = ['accuracy'])
```

In [55]:
```
# setup checkpoint
# Set the path to the folder where you want to save the checkpoint
checkpoint_path2 = "/content/drive/MyDrive/Colab_Checkpoints/checkpoint2.ckpt"

# Save the model's weights and optimizer state
model2.save_weights(checkpoint_path2)



early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.0005, patience=10, mode

checkpoint2 = tf.keras.callbacks.ModelCheckpoint(checkpoint_path2,
                    monitor="val_accuracy", mode="max",
                    save_best_only=True, verbose=1)
```

In [56]:
```
# Fitting the model
history2 = model2.fit(
    train_set,
    validation_data = validation_set,
    callbacks=[early_stopping, checkpoint2],
    epochs = 100)
```

```
Epoch 1/100
471/471 [==============================] - ETA: 0s - loss: 1.6932 - accuracy: 0.2624
Epoch 1: val_accuracy improved from -inf to 0.31120, saving model to /content/drive/M
yDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 24s 42ms/step - loss: 1.6932 - accuracy:
0.2624 - val_loss: 1.3610 - val_accuracy: 0.3112
Epoch 2/100
471/471 [==============================] - ETA: 0s - loss: 1.4826 - accuracy: 0.2642
Epoch 2: val_accuracy did not improve from 0.31120
471/471 [==============================] - 16s 34ms/step - loss: 1.4826 - accuracy:
0.2642 - val_loss: 1.3642 - val_accuracy: 0.2873
Epoch 3/100
471/471 [==============================] - ETA: 0s - loss: 1.4453 - accuracy: 0.2679
Epoch 3: val_accuracy improved from 0.31120 to 0.33273, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.4453 - accuracy:
0.2679 - val_loss: 1.3538 - val_accuracy: 0.3327
Epoch 4/100
471/471 [==============================] - ETA: 0s - loss: 1.4199 - accuracy: 0.2775
Epoch 4: val_accuracy did not improve from 0.33273
471/471 [==============================] - 16s 34ms/step - loss: 1.4199 - accuracy:
0.2775 - val_loss: 1.3592 - val_accuracy: 0.3203
Epoch 5/100
471/471 [==============================] - ETA: 0s - loss: 1.4090 - accuracy: 0.2669
Epoch 5: val_accuracy improved from 0.33273 to 0.34319, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 1.4090 - accuracy:
0.2669 - val_loss: 1.3596 - val_accuracy: 0.3432
Epoch 6/100
470/471 [============================>.] - ETA: 0s - loss: 1.3968 - accuracy: 0.2760
Epoch 6: val_accuracy improved from 0.34319 to 0.34420, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.3968 - accuracy:
0.2759 - val_loss: 1.3572 - val_accuracy: 0.3442
Epoch 7/100
469/471 [============================>.] - ETA: 0s - loss: 1.3909 - accuracy: 0.2802
Epoch 7: val_accuracy improved from 0.34420 to 0.35144, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.3911 - accuracy:
0.2802 - val_loss: 1.3551 - val_accuracy: 0.3514
Epoch 8/100
471/471 [==============================] - ETA: 0s - loss: 1.3880 - accuracy: 0.2765
Epoch 8: val_accuracy did not improve from 0.35144
471/471 [==============================] - 16s 33ms/step - loss: 1.3880 - accuracy:
0.2765 - val_loss: 1.3535 - val_accuracy: 0.3454
Epoch 9/100
471/471 [==============================] - ETA: 0s - loss: 1.3817 - accuracy: 0.2832
Epoch 9: val_accuracy improved from 0.35144 to 0.35767, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.3817 - accuracy:
0.2832 - val_loss: 1.3553 - val_accuracy: 0.3577
Epoch 10/100
470/471 [============================>.] - ETA: 0s - loss: 1.3811 - accuracy: 0.2720
Epoch 10: val_accuracy did not improve from 0.35767
471/471 [==============================] - 16s 33ms/step - loss: 1.3811 - accuracy:
0.2722 - val_loss: 1.3538 - val_accuracy: 0.3380
Epoch 11/100
470/471 [============================>.] - ETA: 0s - loss: 1.3767 - accuracy: 0.2803
Epoch 11: val_accuracy improved from 0.35767 to 0.36733, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
```

```
471/471 [==============================] - 18s 39ms/step - loss: 1.3768 - accuracy:
0.2804 - val_loss: 1.3618 - val_accuracy: 0.3673
Epoch 12/100
471/471 [==============================] - ETA: 0s - loss: 1.3716 - accuracy: 0.2894
Epoch 12: val_accuracy did not improve from 0.36733
471/471 [==============================] - 16s 33ms/step - loss: 1.3716 - accuracy:
0.2894 - val_loss: 1.3533 - val_accuracy: 0.3464
Epoch 13/100
470/471 [============================>.] - ETA: 0s - loss: 1.3628 - accuracy: 0.3023
Epoch 13: val_accuracy did not improve from 0.36733
471/471 [==============================] - 16s 34ms/step - loss: 1.3628 - accuracy:
0.3023 - val_loss: 1.3442 - val_accuracy: 0.3661
Epoch 14/100
471/471 [==============================] - ETA: 0s - loss: 1.3441 - accuracy: 0.3206
Epoch 14: val_accuracy did not improve from 0.36733
471/471 [==============================] - 16s 34ms/step - loss: 1.3441 - accuracy:
0.3206 - val_loss: 1.3281 - val_accuracy: 0.3440
Epoch 15/100
471/471 [==============================] - ETA: 0s - loss: 1.3073 - accuracy: 0.3517
Epoch 15: val_accuracy improved from 0.36733 to 0.37558, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.3073 - accuracy:
0.3517 - val_loss: 1.2554 - val_accuracy: 0.3756
Epoch 16/100
470/471 [============================>.] - ETA: 0s - loss: 1.2815 - accuracy: 0.3666
Epoch 16: val_accuracy did not improve from 0.37558
471/471 [==============================] - 16s 33ms/step - loss: 1.2813 - accuracy:
0.3667 - val_loss: 1.2660 - val_accuracy: 0.3420
Epoch 17/100
471/471 [==============================] - ETA: 0s - loss: 1.2349 - accuracy: 0.3984
Epoch 17: val_accuracy did not improve from 0.37558
471/471 [==============================] - 16s 34ms/step - loss: 1.2349 - accuracy:
0.3984 - val_loss: 1.3339 - val_accuracy: 0.3293
Epoch 18/100
470/471 [============================>.] - ETA: 0s - loss: 1.2000 - accuracy: 0.4180
Epoch 18: val_accuracy improved from 0.37558 to 0.42185, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.2001 - accuracy:
0.4181 - val_loss: 1.1682 - val_accuracy: 0.4218
Epoch 19/100
469/471 [============================>.] - ETA: 0s - loss: 1.1764 - accuracy: 0.4438
Epoch 19: val_accuracy did not improve from 0.42185
471/471 [==============================] - 16s 34ms/step - loss: 1.1765 - accuracy:
0.4435 - val_loss: 1.2229 - val_accuracy: 0.3263
Epoch 20/100
470/471 [============================>.] - ETA: 0s - loss: 1.1531 - accuracy: 0.4662
Epoch 20: val_accuracy improved from 0.42185 to 0.47013, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.1527 - accuracy:
0.4666 - val_loss: 1.1327 - val_accuracy: 0.4701
Epoch 21/100
471/471 [==============================] - ETA: 0s - loss: 1.1225 - accuracy: 0.4935
Epoch 21: val_accuracy improved from 0.47013 to 0.51599, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.1225 - accuracy:
0.4935 - val_loss: 1.0900 - val_accuracy: 0.5160
Epoch 22/100
471/471 [==============================] - ETA: 0s - loss: 1.0847 - accuracy: 0.5176
Epoch 22: val_accuracy improved from 0.51599 to 0.54617, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
```

```
471/471 [==============================] - 18s 38ms/step - loss: 1.0847 - accuracy: 
0.5176 - val_loss: 1.0322 - val_accuracy: 0.5462
Epoch 23/100
471/471 [==============================] - ETA: 0s - loss: 1.0474 - accuracy: 0.5460
Epoch 23: val_accuracy did not improve from 0.54617
471/471 [==============================] - 16s 33ms/step - loss: 1.0474 - accuracy: 
0.5460 - val_loss: 1.1570 - val_accuracy: 0.4299
Epoch 24/100
471/471 [==============================] - ETA: 0s - loss: 1.0331 - accuracy: 0.5544
Epoch 24: val_accuracy improved from 0.54617 to 0.60893, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.0331 - accuracy: 
0.5544 - val_loss: 0.9327 - val_accuracy: 0.6089
Epoch 25/100
470/471 [============================>.] - ETA: 0s - loss: 1.0066 - accuracy: 0.5691
Epoch 25: val_accuracy did not improve from 0.60893
471/471 [==============================] - 16s 34ms/step - loss: 1.0065 - accuracy: 
0.5691 - val_loss: 0.9938 - val_accuracy: 0.5719
Epoch 26/100
471/471 [==============================] - ETA: 0s - loss: 0.9921 - accuracy: 0.5832
Epoch 26: val_accuracy did not improve from 0.60893
471/471 [==============================] - 16s 34ms/step - loss: 0.9921 - accuracy: 
0.5832 - val_loss: 0.9866 - val_accuracy: 0.5723
Epoch 27/100
470/471 [============================>.] - ETA: 0s - loss: 0.9703 - accuracy: 0.5939
Epoch 27: val_accuracy did not improve from 0.60893
471/471 [==============================] - 16s 34ms/step - loss: 0.9701 - accuracy: 
0.5941 - val_loss: 0.9493 - val_accuracy: 0.5951
Epoch 28/100
471/471 [==============================] - ETA: 0s - loss: 0.9630 - accuracy: 0.5993
Epoch 28: val_accuracy improved from 0.60893 to 0.64051, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 0.9630 - accuracy: 
0.5993 - val_loss: 0.8631 - val_accuracy: 0.6405
Epoch 29/100
470/471 [============================>.] - ETA: 0s - loss: 0.9462 - accuracy: 0.6072
Epoch 29: val_accuracy did not improve from 0.64051
471/471 [==============================] - 16s 34ms/step - loss: 0.9463 - accuracy: 
0.6069 - val_loss: 0.9961 - val_accuracy: 0.5633
Epoch 30/100
470/471 [============================>.] - ETA: 0s - loss: 0.9328 - accuracy: 0.6122
Epoch 30: val_accuracy did not improve from 0.64051
471/471 [==============================] - 16s 33ms/step - loss: 0.9326 - accuracy: 
0.6122 - val_loss: 0.8909 - val_accuracy: 0.6172
Epoch 31/100
470/471 [============================>.] - ETA: 0s - loss: 0.9351 - accuracy: 0.6147
Epoch 31: val_accuracy did not improve from 0.64051
471/471 [==============================] - 16s 34ms/step - loss: 0.9347 - accuracy: 
0.6150 - val_loss: 0.9824 - val_accuracy: 0.5707
Epoch 32/100
470/471 [============================>.] - ETA: 0s - loss: 0.9163 - accuracy: 0.6218
Epoch 32: val_accuracy improved from 0.64051 to 0.65983, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.9159 - accuracy: 
0.6220 - val_loss: 0.8188 - val_accuracy: 0.6598
Epoch 33/100
471/471 [==============================] - ETA: 0s - loss: 0.9157 - accuracy: 0.6190
Epoch 33: val_accuracy did not improve from 0.65983
471/471 [==============================] - 16s 34ms/step - loss: 0.9157 - accuracy: 
0.6190 - val_loss: 0.8919 - val_accuracy: 0.6140
```

```
Epoch 34/100
470/471 [============================>.] - ETA: 0s - loss: 0.9023 - accuracy: 0.6297
Epoch 34: val_accuracy improved from 0.65983 to 0.66405, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.9024 - accuracy:
0.6297 - val_loss: 0.8054 - val_accuracy: 0.6641
Epoch 35/100
469/471 [============================>.] - ETA: 0s - loss: 0.8898 - accuracy: 0.6335
Epoch 35: val_accuracy did not improve from 0.66405
471/471 [==============================] - 16s 34ms/step - loss: 0.8896 - accuracy:
0.6338 - val_loss: 0.8215 - val_accuracy: 0.6634
Epoch 36/100
471/471 [==============================] - ETA: 0s - loss: 0.8890 - accuracy: 0.6338
Epoch 36: val_accuracy improved from 0.66405 to 0.67089, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.8890 - accuracy:
0.6338 - val_loss: 0.8074 - val_accuracy: 0.6709
Epoch 37/100
471/471 [==============================] - ETA: 0s - loss: 0.8801 - accuracy: 0.6436
Epoch 37: val_accuracy did not improve from 0.67089
471/471 [==============================] - 16s 34ms/step - loss: 0.8801 - accuracy:
0.6436 - val_loss: 1.0796 - val_accuracy: 0.5077
Epoch 38/100
470/471 [============================>.] - ETA: 0s - loss: 0.8788 - accuracy: 0.6417
Epoch 38: val_accuracy improved from 0.67089 to 0.68678, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.8788 - accuracy:
0.6417 - val_loss: 0.7749 - val_accuracy: 0.6868
Epoch 39/100
470/471 [============================>.] - ETA: 0s - loss: 0.8733 - accuracy: 0.6396
Epoch 39: val_accuracy did not improve from 0.68678
471/471 [==============================] - 16s 34ms/step - loss: 0.8734 - accuracy:
0.6394 - val_loss: 0.9301 - val_accuracy: 0.6039
Epoch 40/100
471/471 [==============================] - ETA: 0s - loss: 0.8646 - accuracy: 0.6479
Epoch 40: val_accuracy did not improve from 0.68678
471/471 [==============================] - 16s 34ms/step - loss: 0.8646 - accuracy:
0.6479 - val_loss: 0.8533 - val_accuracy: 0.6508
Epoch 41/100
470/471 [============================>.] - ETA: 0s - loss: 0.8629 - accuracy: 0.6468
Epoch 41: val_accuracy did not improve from 0.68678
471/471 [==============================] - 16s 33ms/step - loss: 0.8624 - accuracy:
0.6472 - val_loss: 0.8005 - val_accuracy: 0.6685
Epoch 42/100
469/471 [============================>.] - ETA: 0s - loss: 0.8657 - accuracy: 0.6475
Epoch 42: val_accuracy did not improve from 0.68678
471/471 [==============================] - 16s 34ms/step - loss: 0.8655 - accuracy:
0.6476 - val_loss: 0.9514 - val_accuracy: 0.6041
Epoch 43/100
470/471 [============================>.] - ETA: 0s - loss: 0.8545 - accuracy: 0.6501
Epoch 43: val_accuracy improved from 0.68678 to 0.68880, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.8542 - accuracy:
0.6502 - val_loss: 0.7646 - val_accuracy: 0.6888
Epoch 44/100
471/471 [==============================] - ETA: 0s - loss: 0.8499 - accuracy: 0.6599
Epoch 44: val_accuracy did not improve from 0.68880
471/471 [==============================] - 16s 33ms/step - loss: 0.8499 - accuracy:
0.6599 - val_loss: 0.8374 - val_accuracy: 0.6500
Epoch 45/100
```

```
470/471 [============================>.] - ETA: 0s - loss: 0.8460 - accuracy: 0.6590
Epoch 45: val_accuracy did not improve from 0.68880
471/471 [==============================] - 16s 34ms/step - loss: 0.8459 - accuracy:
0.6589 - val_loss: 0.8468 - val_accuracy: 0.6419
Epoch 46/100
469/471 [============================>.] - ETA: 0s - loss: 0.8423 - accuracy: 0.6562
Epoch 46: val_accuracy improved from 0.68880 to 0.69101, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 0.8426 - accuracy:
0.6559 - val_loss: 0.7704 - val_accuracy: 0.6910
Epoch 47/100
470/471 [============================>.] - ETA: 0s - loss: 0.8387 - accuracy: 0.6623
Epoch 47: val_accuracy improved from 0.69101 to 0.70147, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.8389 - accuracy:
0.6619 - val_loss: 0.7505 - val_accuracy: 0.7015
Epoch 48/100
469/471 [============================>.] - ETA: 0s - loss: 0.8378 - accuracy: 0.6598
Epoch 48: val_accuracy did not improve from 0.70147
471/471 [==============================] - 16s 34ms/step - loss: 0.8385 - accuracy:
0.6598 - val_loss: 0.8116 - val_accuracy: 0.6735
Epoch 49/100
471/471 [==============================] - ETA: 0s - loss: 0.8348 - accuracy: 0.6612
Epoch 49: val_accuracy did not improve from 0.70147
471/471 [==============================] - 16s 33ms/step - loss: 0.8348 - accuracy:
0.6612 - val_loss: 0.8238 - val_accuracy: 0.6496
Epoch 50/100
470/471 [============================>.] - ETA: 0s - loss: 0.8272 - accuracy: 0.6669
Epoch 50: val_accuracy did not improve from 0.70147
471/471 [==============================] - 16s 34ms/step - loss: 0.8273 - accuracy:
0.6669 - val_loss: 0.7699 - val_accuracy: 0.6876
Epoch 51/100
471/471 [==============================] - ETA: 0s - loss: 0.8221 - accuracy: 0.6653
Epoch 51: val_accuracy did not improve from 0.70147
471/471 [==============================] - 16s 33ms/step - loss: 0.8221 - accuracy:
0.6653 - val_loss: 0.7814 - val_accuracy: 0.6763
Epoch 52/100
471/471 [==============================] - ETA: 0s - loss: 0.8231 - accuracy: 0.6679
Epoch 52: val_accuracy improved from 0.70147 to 0.71032, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.8231 - accuracy:
0.6679 - val_loss: 0.7242 - val_accuracy: 0.7103
Epoch 53/100
470/471 [============================>.] - ETA: 0s - loss: 0.8294 - accuracy: 0.6639
Epoch 53: val_accuracy did not improve from 0.71032
471/471 [==============================] - 16s 34ms/step - loss: 0.8291 - accuracy:
0.6638 - val_loss: 0.8045 - val_accuracy: 0.6715
Epoch 54/100
470/471 [============================>.] - ETA: 0s - loss: 0.8124 - accuracy: 0.6724
Epoch 54: val_accuracy improved from 0.71032 to 0.71515, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.8124 - accuracy:
0.6724 - val_loss: 0.7204 - val_accuracy: 0.7151
Epoch 55/100
471/471 [==============================] - ETA: 0s - loss: 0.8191 - accuracy: 0.6677
Epoch 55: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 33ms/step - loss: 0.8191 - accuracy:
0.6677 - val_loss: 0.7361 - val_accuracy: 0.7107
Epoch 56/100
471/471 [==============================] - ETA: 0s - loss: 0.8224 - accuracy: 0.6704
```

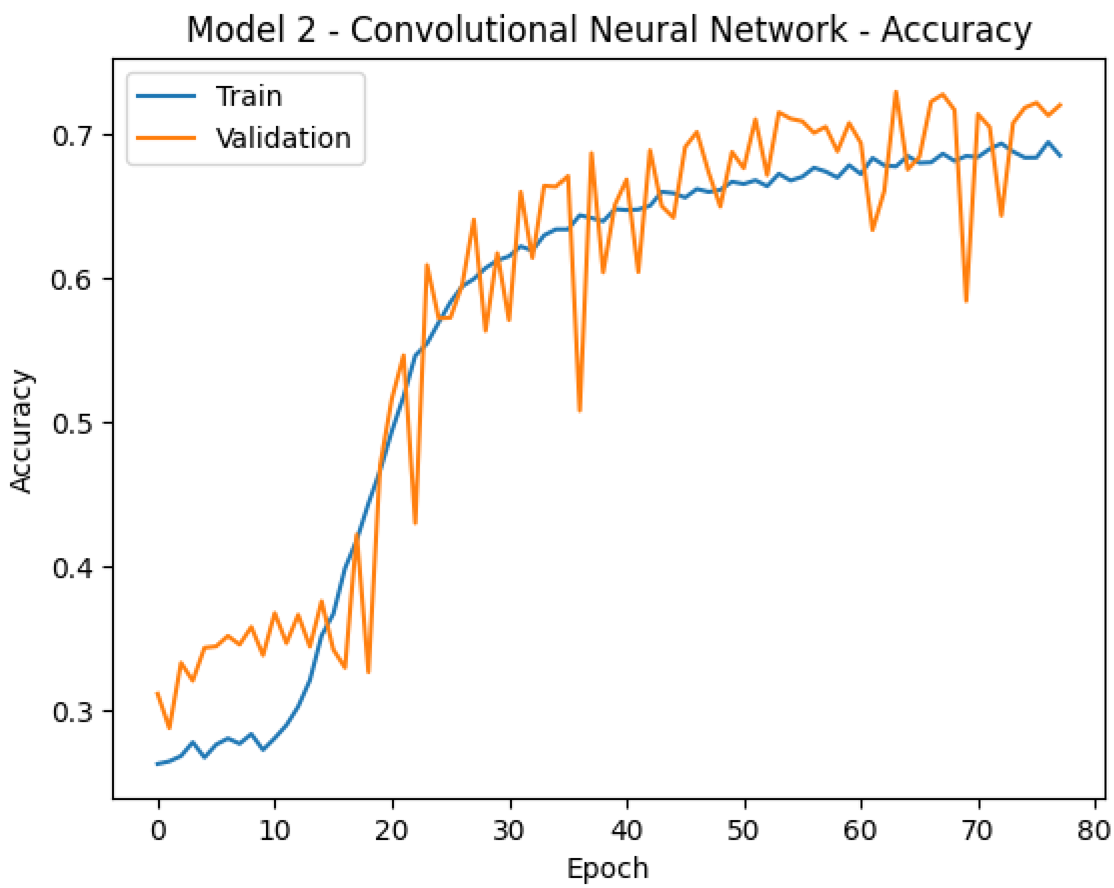
```
Epoch 56: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 33ms/step - loss: 0.8224 - accuracy:
0.6704 - val_loss: 0.7246 - val_accuracy: 0.7089
Epoch 57/100
469/471 [============================>.] - ETA: 0s - loss: 0.8091 - accuracy: 0.6766
Epoch 57: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 34ms/step - loss: 0.8091 - accuracy:
0.6766 - val_loss: 0.7510 - val_accuracy: 0.7009
Epoch 58/100
471/471 [==============================] - ETA: 0s - loss: 0.8179 - accuracy: 0.6739
Epoch 58: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 34ms/step - loss: 0.8179 - accuracy:
0.6739 - val_loss: 0.7265 - val_accuracy: 0.7051
Epoch 59/100
470/471 [============================>.] - ETA: 0s - loss: 0.8138 - accuracy: 0.6699
Epoch 59: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 34ms/step - loss: 0.8138 - accuracy:
0.6698 - val_loss: 0.7542 - val_accuracy: 0.6880
Epoch 60/100
470/471 [============================>.] - ETA: 0s - loss: 0.8038 - accuracy: 0.6784
Epoch 60: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 34ms/step - loss: 0.8034 - accuracy:
0.6784 - val_loss: 0.7198 - val_accuracy: 0.7077
Epoch 61/100
470/471 [============================>.] - ETA: 0s - loss: 0.8048 - accuracy: 0.6723
Epoch 61: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 34ms/step - loss: 0.8050 - accuracy:
0.6722 - val_loss: 0.7477 - val_accuracy: 0.6938
Epoch 62/100
470/471 [============================>.] - ETA: 0s - loss: 0.7919 - accuracy: 0.6834
Epoch 62: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 34ms/step - loss: 0.7922 - accuracy:
0.6834 - val_loss: 0.8870 - val_accuracy: 0.6333
Epoch 63/100
471/471 [==============================] - ETA: 0s - loss: 0.7973 - accuracy: 0.6783
Epoch 63: val_accuracy did not improve from 0.71515
471/471 [==============================] - 16s 34ms/step - loss: 0.7973 - accuracy:
0.6783 - val_loss: 0.8200 - val_accuracy: 0.6604
Epoch 64/100
471/471 [==============================] - ETA: 0s - loss: 0.7942 - accuracy: 0.6775
Epoch 64: val_accuracy improved from 0.71515 to 0.72943, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint2.ckpt
471/471 [==============================] - 18s 38ms/step - loss: 0.7942 - accuracy:
0.6775 - val_loss: 0.6892 - val_accuracy: 0.7294
Epoch 65/100
470/471 [============================>.] - ETA: 0s - loss: 0.7937 - accuracy: 0.6850
Epoch 65: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7943 - accuracy:
0.6848 - val_loss: 0.8036 - val_accuracy: 0.6753
Epoch 66/100
471/471 [==============================] - ETA: 0s - loss: 0.7954 - accuracy: 0.6800
Epoch 66: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7954 - accuracy:
0.6800 - val_loss: 0.7463 - val_accuracy: 0.6842
Epoch 67/100
471/471 [==============================] - ETA: 0s - loss: 0.7901 - accuracy: 0.6804
Epoch 67: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7901 - accuracy:
0.6804 - val_loss: 0.6908 - val_accuracy: 0.7226
Epoch 68/100
```

```
469/471 [============================>.] - ETA: 0s - loss: 0.7827 - accuracy: 0.6860
Epoch 68: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7822 - accuracy:
0.6864 - val_loss: 0.6812 - val_accuracy: 0.7276
Epoch 69/100
471/471 [==============================] - ETA: 0s - loss: 0.7845 - accuracy: 0.6814
Epoch 69: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7845 - accuracy:
0.6814 - val_loss: 0.7019 - val_accuracy: 0.7168
Epoch 70/100
470/471 [============================>.] - ETA: 0s - loss: 0.7894 - accuracy: 0.6848
Epoch 70: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 33ms/step - loss: 0.7894 - accuracy:
0.6848 - val_loss: 0.9694 - val_accuracy: 0.5840
Epoch 71/100
470/471 [============================>.] - ETA: 0s - loss: 0.7891 - accuracy: 0.6842
Epoch 71: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7891 - accuracy:
0.6840 - val_loss: 0.7121 - val_accuracy: 0.7139
Epoch 72/100
470/471 [============================>.] - ETA: 0s - loss: 0.7847 - accuracy: 0.6898
Epoch 72: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7849 - accuracy:
0.6897 - val_loss: 0.7445 - val_accuracy: 0.7049
Epoch 73/100
471/471 [==============================] - ETA: 0s - loss: 0.7763 - accuracy: 0.6935
Epoch 73: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7763 - accuracy:
0.6935 - val_loss: 0.8215 - val_accuracy: 0.6433
Epoch 74/100
471/471 [==============================] - ETA: 0s - loss: 0.7776 - accuracy: 0.6878
Epoch 74: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7776 - accuracy:
0.6878 - val_loss: 0.7094 - val_accuracy: 0.7077
Epoch 75/100
471/471 [==============================] - ETA: 0s - loss: 0.7824 - accuracy: 0.6836
Epoch 75: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 33ms/step - loss: 0.7824 - accuracy:
0.6836 - val_loss: 0.7104 - val_accuracy: 0.7184
Epoch 76/100
470/471 [============================>.] - ETA: 0s - loss: 0.7870 - accuracy: 0.6836
Epoch 76: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7864 - accuracy:
0.6837 - val_loss: 0.6999 - val_accuracy: 0.7218
Epoch 77/100
471/471 [==============================] - ETA: 0s - loss: 0.7703 - accuracy: 0.6945
Epoch 77: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 33ms/step - loss: 0.7703 - accuracy:
0.6945 - val_loss: 0.7099 - val_accuracy: 0.7129
Epoch 78/100
471/471 [==============================] - ETA: 0s - loss: 0.7758 - accuracy: 0.6851
Epoch 78: val_accuracy did not improve from 0.72943
471/471 [==============================] - 16s 34ms/step - loss: 0.7758 - accuracy:
0.6851 - val_loss: 0.6970 - val_accuracy: 0.7202
```

In [57]:

```
plt.plot(history2.history['accuracy'])

plt.plot(history2.history['val_accuracy'])

plt.title('Model 2 - Convolutional Neural Network - Accuracy')
```

```
plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc = 'upper left')

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



This slightly larger model (model2) performed better than (model1) with no signs of overfitting. ~67%% training accuracy/77% validation accuracy. (We shuffled all of the data in the data generator).

Evaluating the Model on the Test Set

In [58]:
```
#set all of the random seeds
np.random.seed(5)
random.seed(5)
tf.random.set_seed(5)

#evaluate accuracy on test set
accuracy = model2.evaluate(test_set, verbose = 2)
```

```
4/4 - 0s - loss: 0.6479 - accuracy: 0.7578 - 220ms/epoch - 55ms/step
```

In [59]:
```
#code to get pairs of predicted vs. true for the confusion matrix
all_y_pred2 = []
all_y_true = []
```

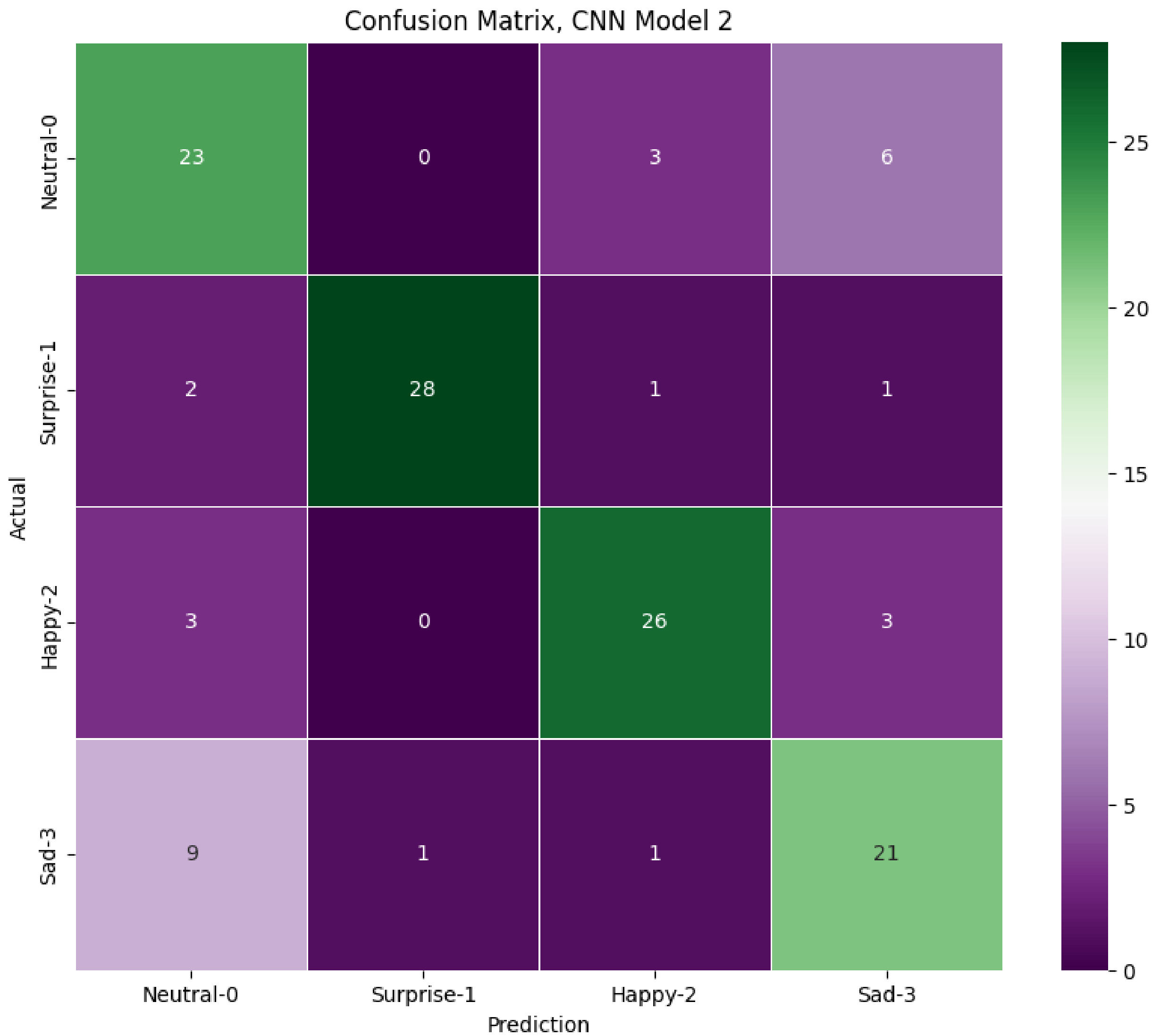
```
for i in range(len(test_set)):
    x , y = test_set[i] #step through each test image/label
    y_pred2 = model2.predict(x) #run image though model, return predicted label

    #all_x.append(x)
    all_y_pred2.append(y_pred2) #add the predicted label to an ordered list
    all_y_true.append(y)      #add the true label to an ordered list

all_y_pred2 = np.concatenate(all_y_pred2, axis=0) #concatenate all preds
all_y_true = np.concatenate(all_y_true, axis=0) #concatenate all base vals


#test_images, test_labels = next(test_set)
#pred = model1.predict(test_images)
pred2 = np.argmax(all_y_pred2, axis=1) #get class indices
y_true = np.argmax(all_y_true, axis=1) #get class indices
```

```
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
```

In [60]:

```
# Printing the classification report
#importing function from sklearn
from sklearn.metrics import classification_report
print(classification_report(y_true, pred2))

# Plotting the Confusion Matrix using confusion matrix() function which is also predef
confusion_matrix2 = tf.math.confusion_matrix(y_true,pred2)
f, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(
    confusion_matrix2,
    annot=True,
    linewidths=.4,
    fmt="d",
    square=True,
    ax=ax,
    cmap="PRGn",
    xticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3'], #added labels for c
    yticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3'] #added labels for cl
)
plt.xlabel('Prediction',fontsize=10)
plt.ylabel('Actual',fontsize=10)
plt.title('Confusion Matrix, CNN Model 2', fontsize=12)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.62      0.72      0.67        32
           1       0.97      0.88      0.92        32
           2       0.84      0.81      0.83        32
           3       0.68      0.66      0.67        32

    accuracy                           0.77       128
   macro avg       0.78      0.77      0.77       128
weighted avg       0.78      0.77      0.77       128
```



In [126…

```
#save model2
model2.save("facialemotionmodel2.h5py")
```

Observations and Insights:

Happy or Surprised:

The model provides clear discernment of "surprised" and "happy". As such, one application for this model: reviews screen captures of reactions to unexpected news or events in suspect identification (suspicious homicides, etc...).

The classification with the most reliable results is "Class 1: surprised" with an F1 score of 92%, and a precision of 97%. We are less likely to get false positives of surprised here; and if the model classifies the instance as "surprised", it likely is. This is unsurprising, given that there are clear indicators of the *surprised* emotion - specifically, wide open mouth and eyes.

Classification reliability "Class 2: Happy" improved, with an acceptable F1 score of 83%. Recall is relatively high at 81%; so the model found 14% of the actual happy, while 19% were actually classified as something else. Precision is 84%, so the model made "happy" predictions somewhat conservatively - not many other instances incorrectly identified as happy (predominately confused with "neutral").

Sad and neutral: How many times have you ever asked someone "What's wrong?" or "What's going on?", and they replied, "Nothing, there's nothing wrong, why do you ask?" It's not always easy to read emotions solely from Facial expressions, because emotions can be mixed (or more reserved), and faces can reflect that. It is more often seen as appropriate to neutralize the expression of "sad" or "negative" emotions, so might be why we see a large interplay between the labeling and -subsequently- the prediction the "neutral", and the "sad" expression.

Applications of sad/neutral: If people only had the four emotions studied here, this model would be fine for use in "conversational" robot-human interaction given that "What's wrong", "What's going on?" or "Is something bothering you?" are all perfectly acceptable (and human-like!) prompts to use in order to glean more information from a respondent. Furthermore, when prompted, the human could share more defineable facial visual cues (or biometric cues) that would enable easier classification by the model. In fact, a sense of "privacy" that is not violated by overly-informed robots could help speed adoption of the technology by invoking a more human experience of robot-human interactions.

classes:
neutral 0
surprise 1
happy 2
sad 3

Conclusion:

The best performing model of all tested is CNN Model Two (2, the second one). This model is about 75%-77% accurate. This model is complex and is very good at SEPARATING instances of the happy (2) and surprised (1) classes from the other classes. There still appears to be a high amount of error in predicting Classes Zero (0 - NEUTRAL) and Three (3 - SAD).

Model Two (2) is preferred over Model Three (3) - see appendix for Model Three (3) - because while both have the same overall accuracy (77%), Model Two (2) maintains a clearer definition between higher number of classes and maintains a higher level of accuracy for a larger number of classes.

The highest accuracy of all of the models is stuck in the 75% region- seemingly due to confusion with the sad and neutral classes. Therefore, a much closer examination of training images classified as such is necessary. Training images of the sad (3) class are very diverse and contain tears, hands, and many different facial positions and expressions, so this class might be tougher to classify due to its diversity and complexity. The neutral (0) class can at times be confused with less emphatic expressions of the remaining three classes.

Other problems with model training are obviously due to the fine line of distinction between neutral and sad. The other classes (HAPPY and SURPRISED) are rarely misclassified as "other classes", **but it seems that "sad" (class 3) and "neutral" (class 4) are often misclassified;** and looking at the images, it is easy to tell why these classes would have a high level of inaccuracy. **Sometimes a bored or neutral face may be misinterpreted as disinterested, sad, or depressed. It's questionable whether it is advisable to delete more photos in order to create an obvious differentiation** between the two emotions, **or simply to train a separate model to detect the finer details that might separate sad from neutral.**

Moving forward, all data sets should be reviewed by parties with some stake in interpreting and labelling expressions from each CLASSES. The labeling of data sets is very important. LIKEWISE, the establishment of appropriate classes and subsequent placement of instances is crucial as well. "Bored", "disgusted", "angry", and other sentiments should be separated from "happy", "sad", "neutral", and "surprised". Classification should be clearly separated to provide a higher level accuracy. Even a class named "Mixed Emotions" or "Ambiguous" or "Hiding Something", would be an improvement over ambivalent classification.

For instances that are very similar in structure and where the variation between images is very slight, larger model architectures are essential.

Training models will still require extended training periods and/or more powerful processors to learn how to properly classify images that have subtle differences.

Refined insights:

- What are the most meaningful insights from the data relevant to the problem?

This classification problem will not be an easy one. The features that are important and/or germane to categorizing by *type of thing* (for example, dog, cat, car, etc...) are a bit different from the features that are necessary to differentiate between *states* of the same thing (hungry cat, excited cat, etc...). **Classification in this sense, can be very subjective. There might not even be any ground truth because sometime people don't even know how they feel. Couple this with the fact that people can have MULTIPLE FEELINGS AT ONCE AND THEIR FACES WILL REFLECT THAT.** Due to such subjectivity, the training data may be labeled inconsistently, especially when the labeling is being done IN PARTS and by more than one person.

Comparison of various techniques and their relative performance:

- How do different techniques perform? Which one is performing relatively better? Is there scope to improve the performance further?

RGB gave slower results, with slightly lower results and was prone to overtraining

Proposal for the final solution design:

- Adopt CNN Model 2 for now. CNN model 2 is basically model 3 with and additional drop layer and minor tweaks of some of the hyperparameters.

(For the final submission, I have added more hyperparameters to the tuner, an additioanl model with 6 convolutional blocks, and I have made minor changes to the data image generator (shuffle=true for all sets)).

Appendix

Transfer Learning Architectures

Three Transfer Learning architectures were evaluated in the milestone phase of this project. For the sake of brevity, all code and discussion of the transfer learning architectures was deleted from this workbook.

When using transfer learning architectures to train new models on different data sets, it is essential to use a learning architecture that is trained on similar training images OR to have a good grasp of how the hidden layers within the transfer architecture are defining the task. For example, convolution layers 1-3 of architecture X may provide a great basis for defining edges and contours, but beyond those layers, they may be better at identifying different distinguishing characteristics.

These transfer architectures did not yield models that improved on the accuracy of our convolutional layer networks built from scratch.

Building Complex Neural Network Architectures; Models 3 and 4

In this section, we will build a more complex Convolutional Neural Network Model that has close to as many parameters as we had in our Transfer Learning Models. However, we will have only 1 input channel for our input images.

Creating our Data Loaders

In this section, we are creating data loaders which we will use as inputs to the more Complicated Convolutional Neural Network. We will go ahead with color_mode = 'grayscale'.

In [61]:
```
# Clearing backend

backend.clear_session()

#set random seed
np.random.seed(5)
```

```
random.seed(5)
tf.random.set_seed(5)
```

In [62]:

```
# the data that the models use will be passed through this data augmentation process

# Fixing the seed for random number generators
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
seed=42 #defining seed for random generators

batch_size  = 32

datagen_train  = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    height_shift_range=0.2,
    width_shift_range=0.2,
    brightness_range=(0.5,1.5),
    shear_range = 0.2,
    rotation_range= 60)

datagen_val = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    height_shift_range=0.2,
    width_shift_range=0.2,
    brightness_range=(0.5,1.5),
    shear_range = 0.2,
    rotation_range= 60)

datagen_test = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    height_shift_range=0.2,
    width_shift_range=0.2,
    brightness_range=(0.5,1.5),
    shear_range = 0.2,
    rotation_range= 60)

train_set = datagen_train.flow_from_directory(where_is+"train",
                                              target_size = (ts, ts),
                                              color_mode = "grayscale",
                                              batch_size = batch_size,
                                              class_mode = 'categorical',
                                              classes = classes,
                                              seed=seed, #added random seed setting fo
                                              shuffle = True)

validation_set = datagen_val.flow_from_directory(where_is + 'validation/',
                                              target_size = (ts,ts),
                                              color_mode = "grayscale",
                                              batch_size = batch_size,
                                              class_mode = 'categorical',
                                              classes = classes,
                                              seed=seed,
                                              shuffle = True)

test_set = datagen_test.flow_from_directory(where_is + 'test/',
```

```
                                                  target_size = (ts,ts),
                                                  color_mode = "grayscale",
                                                  batch_size = batch_size,
                                                  class_mode = 'categorical',
                                                  classes = classes,
                                                  seed=seed,
                                                  shuffle = True)
```

```
Found 15059 images belonging to 4 classes.
Found 4971 images belonging to 4 classes.
Found 128 images belonging to 4 classes.
```

Model Building: Variations on 5 Convolutional Blocks

- Try building a layer with 5 Convolutional Blocks and see if performance increases.

Model 2 already had 5 convolutional blocks, so we adjusted the kernel size and drop rate. Then we deleted the final drop layer between the last fully connected (FC) layer and the classification layer. We also adjusted the filters, activation, and learning rate by using an alternate trial from Model 2's tuning exercise. The resulting five Convolutional block model (Model 3) improved our overall accuracy and greatly increased the useability of the model by refining greatly increasing the accuracy for Classes 1 and 2 (Surprised and happy).

In [63]:
```
drop=0.2

#setting parameters
activation1 = LeakyReLU(alpha=0.1)
#optimizer = SGD(learning_rate=0.0010 , momentum=0.9)
optimizer = Adam(learning_rate =0.002317701860832247)
learning_rate= 0.002317701860832247
units1= 512
units2= 416
units3= 416
units4= 64
units5= 352
unitsfc1= 128
unitsfc2= 256
unitsfc3= 224
#Score: 0.32890766859054565


#initialize model
model3 = Sequential()

# First Convolutional layer with 32 filters and the kernel size of 3x3. Use the 'same'
model3.add(Conv2D(units1, kernel_size=(3, 3), input_shape = (48, 48, 1), padding = 'sa
#Max Pooling
model3.add(MaxPooling2D(2, 2))

# Second Convolutional Block
model3.add(Conv2D(units2, kernel_size=(2, 2), padding = 'same', activation=activation1
#Max Pooling
model3.add(MaxPooling2D(2, 2))
#Add a dropout layer
model3.add(Dropout(drop))
#Add a BatchNormalization layer
```

```
model3.add(BatchNormalization())

# Third Convolutional Block
model3.add(Conv2D(units3, kernel_size=(2, 2), padding = 'same', activation=activation1
#Max Pooling
model3.add(MaxPooling2D(2, 2))
#Add a dropout layer
model3.add(Dropout(drop))
#Add a BatchNormalization layer
model3.add(BatchNormalization())

# Fourth Convolutional Block
model3.add(Conv2D(units4, kernel_size=(2, 2), padding = 'same', activation=activation1
#Max Pooling
model3.add(MaxPooling2D(2, 2))
#Add a dropout layer
model3.add(Dropout(drop))
#Add a BatchNormalization layer
model3.add(BatchNormalization())

# Fifth Convolutional Block, default activation relu
model3.add(Conv2D(units5, kernel_size=(2, 2), padding = 'same')) #256
#Max Pooling
model3.add(MaxPooling2D(2, 2))
#Add a dropout layer
model3.add(Dropout(drop))
#Add a BatchNormalization layer
model3.add(BatchNormalization())

#Flatten for dense layers
model3.add(Flatten())

# First fully Connected Block
#default activation is relu
model3.add(Dense(unitsfc1)) #512
#Add a dropout layer
model3.add(Dropout(drop))

# Second fully Connected Block
model3.add(Dense(unitsfc2)) #256
#Add a dropout layer
model3.add(Dropout(drop))

# Third fully Connected Block
model3.add(Dense(unitsfc3)) #128

#Add a dropout layer
#model3.add(Dropout(drop))


# Classifier Block
model3.add(Dense(4, activation = 'softmax'))
loss_function = 'categorical_crossentropy'

model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 512)       5120      
                                                                 
 max_pooling2d (MaxPooling2  (None, 24, 24, 512)       0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 24, 24, 416)       852384    
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 416)       0         
 g2D)                                                            
                                                                 
 dropout (Dropout)           (None, 12, 12, 416)       0         
                                                                 
 batch_normalization (Batch  (None, 12, 12, 416)       1664      
 Normalization)                                                  
                                                                 
 conv2d_2 (Conv2D)           (None, 12, 12, 416)       692640    
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 416)         0         
 g2D)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 6, 6, 416)         0         
                                                                 
 batch_normalization_1 (Bat  (None, 6, 6, 416)         1664      
 chNormalization)                                                
                                                                 
 conv2d_3 (Conv2D)           (None, 6, 6, 64)          106560    
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 3, 3, 64)          0         
 g2D)                                                            
                                                                 
 dropout_2 (Dropout)         (None, 3, 3, 64)          0         
                                                                 
 batch_normalization_2 (Bat  (None, 3, 3, 64)          256       
 chNormalization)                                                
                                                                 
 conv2d_4 (Conv2D)           (None, 3, 3, 352)         90464     
                                                                 
 max_pooling2d_4 (MaxPoolin  (None, 1, 1, 352)         0         
 g2D)                                                            
                                                                 
 dropout_3 (Dropout)         (None, 1, 1, 352)         0         
                                                                 
 batch_normalization_3 (Bat  (None, 1, 1, 352)         1408      
 chNormalization)                                                
                                                                 
 flatten (Flatten)           (None, 352)               0         
                                                                 
 dense (Dense)               (None, 128)               45184     
                                                                 
 dropout_4 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 256)               33024     
                                                                 
 dropout_5 (Dropout)         (None, 256)               0         
                                                                 
 dense_2 (Dense)             (None, 224)               57568     
```

```
 dense_3 (Dense)             (None, 4)                 900

=================================================================
Total params: 1888836 (7.21 MB)
Trainable params: 1886340 (7.20 MB)
Non-trainable params: 2496 (9.75 KB)
_________________________________________________________________
```

In [64]:

```
# Compiling the model
model3.compile(loss = loss_function, optimizer = optimizer , metrics = ['accuracy'])
```

In [65]:

```
# setup checkpoint for early stopping
# Set the path to the folder where you want to save the checkpoint
checkpoint_path3 = "/content/drive/MyDrive/Colab_Checkpoints/checkpoint3.ckpt"

# Save the model's weights and optimizer state
model3.save_weights(checkpoint_path3)



early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.0005, patience=10, mode

#changed checkpoint parameters to save the model that gives the optimal val_accuracy
checkpoint3 = tf.keras.callbacks.ModelCheckpoint(checkpoint_path3,
                    monitor="val_accuracy", mode="max",
                    save_best_only=True, verbose=1)
```

In [66]:

```
# Fitting the model
history3 = model3.fit(
    train_set,
    validation_data = validation_set,
    callbacks=[early_stopping, checkpoint3],
    epochs = 100)
```

```
Epoch 1/100
470/471 [============================>.] - ETA: 0s - loss: 1.5791 - accuracy: 0.2658
Epoch 1: val_accuracy improved from -inf to 0.33353, saving model to /content/drive/M
yDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 24s 43ms/step - loss: 1.5787 - accuracy:
0.2656 - val_loss: 1.3663 - val_accuracy: 0.3335
Epoch 2/100
470/471 [============================>.] - ETA: 0s - loss: 1.3923 - accuracy: 0.2828
Epoch 2: val_accuracy did not improve from 0.33353
471/471 [==============================] - 16s 34ms/step - loss: 1.3924 - accuracy:
0.2824 - val_loss: 1.3717 - val_accuracy: 0.3156
Epoch 3/100
471/471 [==============================] - ETA: 0s - loss: 1.3852 - accuracy: 0.2863
Epoch 3: val_accuracy did not improve from 0.33353
471/471 [==============================] - 16s 35ms/step - loss: 1.3852 - accuracy:
0.2863 - val_loss: 1.4068 - val_accuracy: 0.2782
Epoch 4/100
470/471 [============================>.] - ETA: 0s - loss: 1.3742 - accuracy: 0.2991
Epoch 4: val_accuracy did not improve from 0.33353
471/471 [==============================] - 16s 34ms/step - loss: 1.3742 - accuracy:
0.2990 - val_loss: 1.5231 - val_accuracy: 0.2832
Epoch 5/100
471/471 [==============================] - ETA: 0s - loss: 1.3627 - accuracy: 0.3176
Epoch 5: val_accuracy improved from 0.33353 to 0.36693, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.3627 - accuracy:
0.3176 - val_loss: 1.7493 - val_accuracy: 0.3669
Epoch 6/100
470/471 [============================>.] - ETA: 0s - loss: 1.3377 - accuracy: 0.3351
Epoch 6: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.3376 - accuracy:
0.3351 - val_loss: 1.3733 - val_accuracy: 0.2613
Epoch 7/100
470/471 [============================>.] - ETA: 0s - loss: 1.3089 - accuracy: 0.3592
Epoch 7: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.3091 - accuracy:
0.3591 - val_loss: 1.5463 - val_accuracy: 0.2193
Epoch 8/100
470/471 [============================>.] - ETA: 0s - loss: 1.2790 - accuracy: 0.3777
Epoch 8: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.2790 - accuracy:
0.3774 - val_loss: 1.3063 - val_accuracy: 0.2905
Epoch 9/100
471/471 [==============================] - ETA: 0s - loss: 1.2409 - accuracy: 0.4118
Epoch 9: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.2409 - accuracy:
0.4118 - val_loss: 1.2345 - val_accuracy: 0.3563
Epoch 10/100
470/471 [============================>.] - ETA: 0s - loss: 1.1834 - accuracy: 0.4614
Epoch 10: val_accuracy improved from 0.36693 to 0.39972, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.1832 - accuracy:
0.4617 - val_loss: 1.1852 - val_accuracy: 0.3997
Epoch 11/100
470/471 [============================>.] - ETA: 0s - loss: 1.1232 - accuracy: 0.5007
Epoch 11: val_accuracy improved from 0.39972 to 0.49628, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 1.1227 - accuracy:
0.5010 - val_loss: 1.1440 - val_accuracy: 0.4963
Epoch 12/100
```

```
471/471 [==============================] - ETA: 0s - loss: 1.0688 - accuracy: 0.5346
Epoch 12: val_accuracy improved from 0.49628 to 0.53168, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.0688 - accuracy:
0.5346 - val_loss: 1.0839 - val_accuracy: 0.5317
Epoch 13/100
471/471 [==============================] - ETA: 0s - loss: 1.0344 - accuracy: 0.5526
Epoch 13: val_accuracy did not improve from 0.53168
471/471 [==============================] - 16s 34ms/step - loss: 1.0344 - accuracy:
0.5526 - val_loss: 1.3147 - val_accuracy: 0.4315
Epoch 14/100
471/471 [==============================] - ETA: 0s - loss: 1.0141 - accuracy: 0.5666
Epoch 14: val_accuracy improved from 0.53168 to 0.60712, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 1.0141 - accuracy:
0.5666 - val_loss: 0.9240 - val_accuracy: 0.6071
Epoch 15/100
471/471 [==============================] - ETA: 0s - loss: 0.9894 - accuracy: 0.5759
Epoch 15: val_accuracy did not improve from 0.60712
471/471 [==============================] - 16s 35ms/step - loss: 0.9894 - accuracy:
0.5759 - val_loss: 0.9665 - val_accuracy: 0.6009
Epoch 16/100
470/471 [============================>.] - ETA: 0s - loss: 0.9813 - accuracy: 0.5855
Epoch 16: val_accuracy did not improve from 0.60712
471/471 [==============================] - 16s 34ms/step - loss: 0.9814 - accuracy:
0.5854 - val_loss: 1.0244 - val_accuracy: 0.5538
Epoch 17/100
470/471 [============================>.] - ETA: 0s - loss: 0.9588 - accuracy: 0.5948
Epoch 17: val_accuracy did not improve from 0.60712
471/471 [==============================] - 16s 34ms/step - loss: 0.9591 - accuracy:
0.5947 - val_loss: 0.9418 - val_accuracy: 0.5979
Epoch 18/100
471/471 [==============================] - ETA: 0s - loss: 0.9489 - accuracy: 0.5953
Epoch 18: val_accuracy did not improve from 0.60712
471/471 [==============================] - 16s 34ms/step - loss: 0.9489 - accuracy:
0.5953 - val_loss: 1.0266 - val_accuracy: 0.5484
Epoch 19/100
470/471 [============================>.] - ETA: 0s - loss: 0.9418 - accuracy: 0.6058
Epoch 19: val_accuracy did not improve from 0.60712
471/471 [==============================] - 16s 34ms/step - loss: 0.9420 - accuracy:
0.6060 - val_loss: 1.2787 - val_accuracy: 0.3975
Epoch 20/100
470/471 [============================>.] - ETA: 0s - loss: 0.9336 - accuracy: 0.6048
Epoch 20: val_accuracy did not improve from 0.60712
471/471 [==============================] - 16s 34ms/step - loss: 0.9337 - accuracy:
0.6048 - val_loss: 1.1473 - val_accuracy: 0.4599
Epoch 21/100
471/471 [==============================] - ETA: 0s - loss: 0.9266 - accuracy: 0.6112
Epoch 21: val_accuracy improved from 0.60712 to 0.62442, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.9266 - accuracy:
0.6112 - val_loss: 0.9199 - val_accuracy: 0.6244
Epoch 22/100
471/471 [==============================] - ETA: 0s - loss: 0.9153 - accuracy: 0.6144
Epoch 22: val_accuracy did not improve from 0.62442
471/471 [==============================] - 16s 35ms/step - loss: 0.9153 - accuracy:
0.6144 - val_loss: 0.9363 - val_accuracy: 0.6188
Epoch 23/100
471/471 [==============================] - ETA: 0s - loss: 0.9134 - accuracy: 0.6224
Epoch 23: val_accuracy did not improve from 0.62442
```

```
471/471 [==============================] - 16s 34ms/step - loss: 0.9134 - accuracy:
0.6224 - val_loss: 0.9053 - val_accuracy: 0.6134
Epoch 24/100
471/471 [==============================] - ETA: 0s - loss: 0.9098 - accuracy: 0.6198
Epoch 24: val_accuracy did not improve from 0.62442
471/471 [==============================] - 16s 34ms/step - loss: 0.9098 - accuracy:
0.6198 - val_loss: 0.9443 - val_accuracy: 0.5949
Epoch 25/100
470/471 [============================>.] - ETA: 0s - loss: 0.8959 - accuracy: 0.6239
Epoch 25: val_accuracy did not improve from 0.62442
471/471 [==============================] - 16s 35ms/step - loss: 0.8964 - accuracy:
0.6237 - val_loss: 0.9447 - val_accuracy: 0.5916
Epoch 26/100
470/471 [============================>.] - ETA: 0s - loss: 0.8941 - accuracy: 0.6277
Epoch 26: val_accuracy did not improve from 0.62442
471/471 [==============================] - 16s 34ms/step - loss: 0.8946 - accuracy:
0.6277 - val_loss: 0.9224 - val_accuracy: 0.6154
Epoch 27/100
469/471 [============================>.] - ETA: 0s - loss: 0.8903 - accuracy: 0.6296
Epoch 27: val_accuracy improved from 0.62442 to 0.67391, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.8907 - accuracy:
0.6297 - val_loss: 0.8193 - val_accuracy: 0.6739
Epoch 28/100
470/471 [============================>.] - ETA: 0s - loss: 0.8886 - accuracy: 0.6337
Epoch 28: val_accuracy did not improve from 0.67391
471/471 [==============================] - 16s 34ms/step - loss: 0.8882 - accuracy:
0.6340 - val_loss: 1.0801 - val_accuracy: 0.5802
Epoch 29/100
470/471 [============================>.] - ETA: 0s - loss: 0.8854 - accuracy: 0.6303
Epoch 29: val_accuracy did not improve from 0.67391
471/471 [==============================] - 16s 34ms/step - loss: 0.8856 - accuracy:
0.6300 - val_loss: 0.9767 - val_accuracy: 0.5381
Epoch 30/100
471/471 [==============================] - ETA: 0s - loss: 0.8736 - accuracy: 0.6396
Epoch 30: val_accuracy did not improve from 0.67391
471/471 [==============================] - 16s 34ms/step - loss: 0.8736 - accuracy:
0.6396 - val_loss: 0.9537 - val_accuracy: 0.5975
Epoch 31/100
470/471 [============================>.] - ETA: 0s - loss: 0.8750 - accuracy: 0.6359
Epoch 31: val_accuracy did not improve from 0.67391
471/471 [==============================] - 16s 34ms/step - loss: 0.8755 - accuracy:
0.6359 - val_loss: 0.8630 - val_accuracy: 0.6337
Epoch 32/100
470/471 [============================>.] - ETA: 0s - loss: 0.8729 - accuracy: 0.6386
Epoch 32: val_accuracy improved from 0.67391 to 0.67411, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 0.8726 - accuracy:
0.6390 - val_loss: 0.8202 - val_accuracy: 0.6741
Epoch 33/100
470/471 [============================>.] - ETA: 0s - loss: 0.8734 - accuracy: 0.6412
Epoch 33: val_accuracy did not improve from 0.67411
471/471 [==============================] - 16s 34ms/step - loss: 0.8734 - accuracy:
0.6410 - val_loss: 0.8112 - val_accuracy: 0.6711
Epoch 34/100
471/471 [==============================] - ETA: 0s - loss: 0.8694 - accuracy: 0.6384
Epoch 34: val_accuracy did not improve from 0.67411
471/471 [==============================] - 16s 34ms/step - loss: 0.8694 - accuracy:
0.6384 - val_loss: 0.8448 - val_accuracy: 0.6433
Epoch 35/100
```

```
470/471 [============================>.] - ETA: 0s - loss: 0.8622 - accuracy: 0.6424
Epoch 35: val_accuracy did not improve from 0.67411
471/471 [==============================] - 16s 34ms/step - loss: 0.8622 - accuracy:
0.6423 - val_loss: 0.9644 - val_accuracy: 0.5685
Epoch 36/100
470/471 [============================>.] - ETA: 0s - loss: 0.8673 - accuracy: 0.6373
Epoch 36: val_accuracy did not improve from 0.67411
471/471 [==============================] - 16s 35ms/step - loss: 0.8673 - accuracy:
0.6372 - val_loss: 0.8990 - val_accuracy: 0.6079
Epoch 37/100
471/471 [==============================] - ETA: 0s - loss: 0.8617 - accuracy: 0.6475
Epoch 37: val_accuracy did not improve from 0.67411
471/471 [==============================] - 16s 34ms/step - loss: 0.8617 - accuracy:
0.6475 - val_loss: 0.8559 - val_accuracy: 0.6524
Epoch 38/100
470/471 [============================>.] - ETA: 0s - loss: 0.8592 - accuracy: 0.6436
Epoch 38: val_accuracy did not improve from 0.67411
471/471 [==============================] - 16s 34ms/step - loss: 0.8590 - accuracy:
0.6436 - val_loss: 0.8005 - val_accuracy: 0.6721
Epoch 39/100
471/471 [==============================] - ETA: 0s - loss: 0.8507 - accuracy: 0.6504
Epoch 39: val_accuracy improved from 0.67411 to 0.67894, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.8507 - accuracy:
0.6504 - val_loss: 0.7835 - val_accuracy: 0.6789
Epoch 40/100
470/471 [============================>.] - ETA: 0s - loss: 0.8481 - accuracy: 0.6493
Epoch 40: val_accuracy did not improve from 0.67894
471/471 [==============================] - 16s 35ms/step - loss: 0.8486 - accuracy:
0.6489 - val_loss: 0.8380 - val_accuracy: 0.6457
Epoch 41/100
471/471 [==============================] - ETA: 0s - loss: 0.8478 - accuracy: 0.6475
Epoch 41: val_accuracy did not improve from 0.67894
471/471 [==============================] - 16s 34ms/step - loss: 0.8478 - accuracy:
0.6475 - val_loss: 0.8220 - val_accuracy: 0.6474
Epoch 42/100
469/471 [============================>.] - ETA: 0s - loss: 0.8376 - accuracy: 0.6528
Epoch 42: val_accuracy did not improve from 0.67894
471/471 [==============================] - 16s 34ms/step - loss: 0.8368 - accuracy:
0.6534 - val_loss: 1.0285 - val_accuracy: 0.5401
Epoch 43/100
470/471 [============================>.] - ETA: 0s - loss: 0.8420 - accuracy: 0.6552
Epoch 43: val_accuracy did not improve from 0.67894
471/471 [==============================] - 16s 34ms/step - loss: 0.8416 - accuracy:
0.6553 - val_loss: 0.8047 - val_accuracy: 0.6719
Epoch 44/100
470/471 [============================>.] - ETA: 0s - loss: 0.8409 - accuracy: 0.6516
Epoch 44: val_accuracy did not improve from 0.67894
471/471 [==============================] - 16s 34ms/step - loss: 0.8404 - accuracy:
0.6517 - val_loss: 0.9023 - val_accuracy: 0.6168
Epoch 45/100
470/471 [============================>.] - ETA: 0s - loss: 0.8326 - accuracy: 0.6581
Epoch 45: val_accuracy did not improve from 0.67894
471/471 [==============================] - 16s 34ms/step - loss: 0.8328 - accuracy:
0.6581 - val_loss: 0.8052 - val_accuracy: 0.6530
Epoch 46/100
471/471 [==============================] - ETA: 0s - loss: 0.8317 - accuracy: 0.6587
Epoch 46: val_accuracy improved from 0.67894 to 0.69543, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.8317 - accuracy:
```

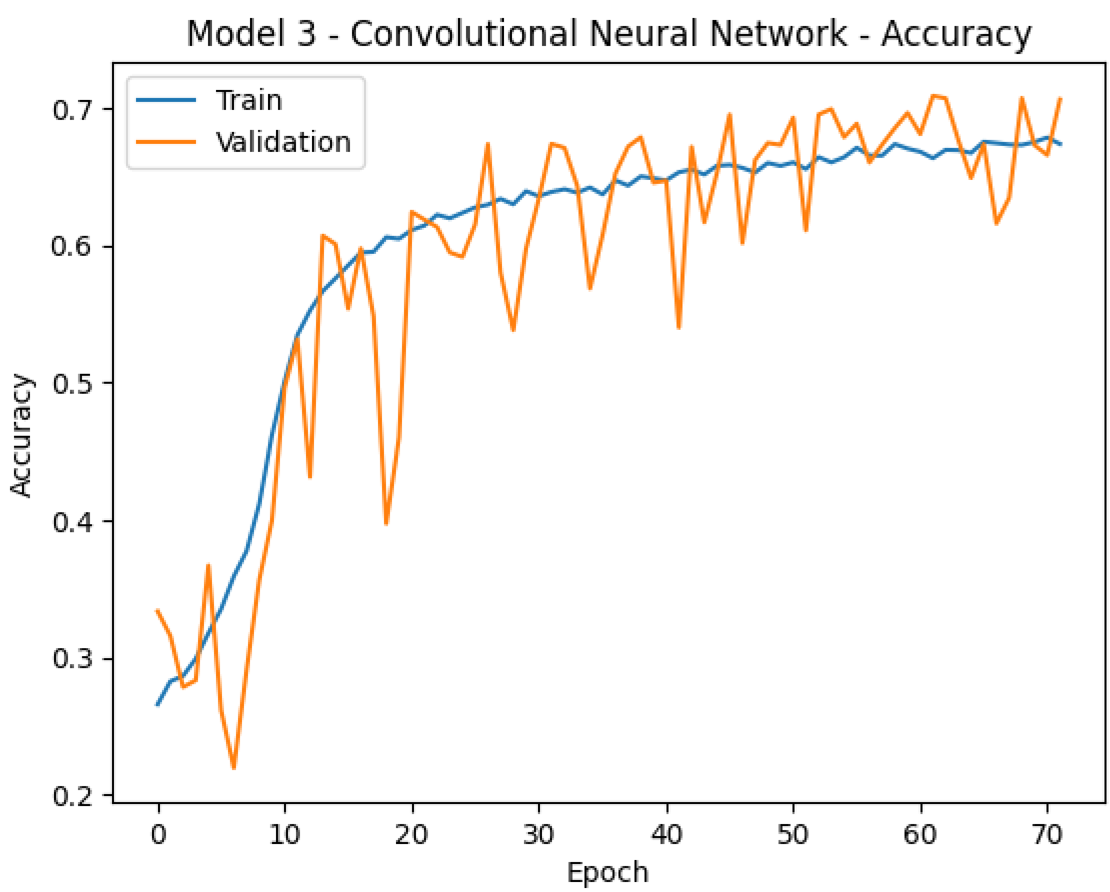
```
0.6587 - val_loss: 0.7669 - val_accuracy: 0.6954
Epoch 47/100
471/471 [==============================] - ETA: 0s - loss: 0.8317 - accuracy: 0.6566
Epoch 47: val_accuracy did not improve from 0.69543
471/471 [==============================] - 16s 34ms/step - loss: 0.8317 - accuracy:
0.6566 - val_loss: 0.9278 - val_accuracy: 0.6017
Epoch 48/100
471/471 [==============================] - ETA: 0s - loss: 0.8430 - accuracy: 0.6532
Epoch 48: val_accuracy did not improve from 0.69543
471/471 [==============================] - 16s 34ms/step - loss: 0.8430 - accuracy:
0.6532 - val_loss: 0.8037 - val_accuracy: 0.6622
Epoch 49/100
471/471 [==============================] - ETA: 0s - loss: 0.8358 - accuracy: 0.6601
Epoch 49: val_accuracy did not improve from 0.69543
471/471 [==============================] - 16s 34ms/step - loss: 0.8358 - accuracy:
0.6601 - val_loss: 0.7936 - val_accuracy: 0.6747
Epoch 50/100
471/471 [==============================] - ETA: 0s - loss: 0.8297 - accuracy: 0.6580
Epoch 50: val_accuracy did not improve from 0.69543
471/471 [==============================] - 16s 35ms/step - loss: 0.8297 - accuracy:
0.6580 - val_loss: 0.7867 - val_accuracy: 0.6733
Epoch 51/100
469/471 [============================>.] - ETA: 0s - loss: 0.8252 - accuracy: 0.6606
Epoch 51: val_accuracy did not improve from 0.69543
471/471 [==============================] - 16s 34ms/step - loss: 0.8251 - accuracy:
0.6605 - val_loss: 0.7582 - val_accuracy: 0.6932
Epoch 52/100
469/471 [============================>.] - ETA: 0s - loss: 0.8286 - accuracy: 0.6558
Epoch 52: val_accuracy did not improve from 0.69543
471/471 [==============================] - 16s 34ms/step - loss: 0.8286 - accuracy:
0.6557 - val_loss: 0.8944 - val_accuracy: 0.6109
Epoch 53/100
471/471 [==============================] - ETA: 0s - loss: 0.8201 - accuracy: 0.6644
Epoch 53: val_accuracy did not improve from 0.69543
471/471 [==============================] - 16s 34ms/step - loss: 0.8201 - accuracy:
0.6644 - val_loss: 0.7500 - val_accuracy: 0.6954
Epoch 54/100
470/471 [============================>.] - ETA: 0s - loss: 0.8288 - accuracy: 0.6605
Epoch 54: val_accuracy improved from 0.69543 to 0.69946, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.8285 - accuracy:
0.6605 - val_loss: 0.7381 - val_accuracy: 0.6995
Epoch 55/100
470/471 [============================>.] - ETA: 0s - loss: 0.8208 - accuracy: 0.6645
Epoch 55: val_accuracy did not improve from 0.69946
471/471 [==============================] - 16s 34ms/step - loss: 0.8208 - accuracy:
0.6643 - val_loss: 0.7970 - val_accuracy: 0.6789
Epoch 56/100
470/471 [============================>.] - ETA: 0s - loss: 0.8181 - accuracy: 0.6714
Epoch 56: val_accuracy did not improve from 0.69946
471/471 [==============================] - 16s 34ms/step - loss: 0.8185 - accuracy:
0.6712 - val_loss: 0.7634 - val_accuracy: 0.6888
Epoch 57/100
470/471 [============================>.] - ETA: 0s - loss: 0.8202 - accuracy: 0.6655
Epoch 57: val_accuracy did not improve from 0.69946
471/471 [==============================] - 16s 34ms/step - loss: 0.8201 - accuracy:
0.6656 - val_loss: 0.8094 - val_accuracy: 0.6604
Epoch 58/100
470/471 [============================>.] - ETA: 0s - loss: 0.8152 - accuracy: 0.6656
Epoch 58: val_accuracy did not improve from 0.69946
```

```
471/471 [==============================] - 16s 34ms/step - loss: 0.8158 - accuracy:
0.6655 - val_loss: 0.7840 - val_accuracy: 0.6735
Epoch 59/100
470/471 [============================>.] - ETA: 0s - loss: 0.8030 - accuracy: 0.6741
Epoch 59: val_accuracy did not improve from 0.69946
471/471 [==============================] - 16s 35ms/step - loss: 0.8032 - accuracy:
0.6739 - val_loss: 0.7718 - val_accuracy: 0.6856
Epoch 60/100
470/471 [============================>.] - ETA: 0s - loss: 0.8131 - accuracy: 0.6707
Epoch 60: val_accuracy did not improve from 0.69946
471/471 [==============================] - 16s 34ms/step - loss: 0.8131 - accuracy:
0.6706 - val_loss: 0.7490 - val_accuracy: 0.6966
Epoch 61/100
470/471 [============================>.] - ETA: 0s - loss: 0.8147 - accuracy: 0.6684
Epoch 61: val_accuracy did not improve from 0.69946
471/471 [==============================] - 16s 34ms/step - loss: 0.8151 - accuracy:
0.6683 - val_loss: 0.7869 - val_accuracy: 0.6812
Epoch 62/100
470/471 [============================>.] - ETA: 0s - loss: 0.8192 - accuracy: 0.6635
Epoch 62: val_accuracy improved from 0.69946 to 0.70911, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint3.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.8193 - accuracy:
0.6636 - val_loss: 0.7329 - val_accuracy: 0.7091
Epoch 63/100
470/471 [============================>.] - ETA: 0s - loss: 0.8082 - accuracy: 0.6697
Epoch 63: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 34ms/step - loss: 0.8081 - accuracy:
0.6697 - val_loss: 0.7326 - val_accuracy: 0.7073
Epoch 64/100
471/471 [==============================] - ETA: 0s - loss: 0.8045 - accuracy: 0.6694
Epoch 64: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 35ms/step - loss: 0.8045 - accuracy:
0.6694 - val_loss: 0.8067 - val_accuracy: 0.6767
Epoch 65/100
471/471 [==============================] - ETA: 0s - loss: 0.8147 - accuracy: 0.6674
Epoch 65: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 34ms/step - loss: 0.8147 - accuracy:
0.6674 - val_loss: 0.8683 - val_accuracy: 0.6492
Epoch 66/100
471/471 [==============================] - ETA: 0s - loss: 0.8033 - accuracy: 0.6756
Epoch 66: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 35ms/step - loss: 0.8033 - accuracy:
0.6756 - val_loss: 0.7792 - val_accuracy: 0.6739
Epoch 67/100
471/471 [==============================] - ETA: 0s - loss: 0.8040 - accuracy: 0.6746
Epoch 67: val_accuracy did not improve from 0.70911
471/471 [==============================] - 17s 35ms/step - loss: 0.8040 - accuracy:
0.6746 - val_loss: 0.9278 - val_accuracy: 0.6160
Epoch 68/100
470/471 [============================>.] - ETA: 0s - loss: 0.8009 - accuracy: 0.6737
Epoch 68: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 35ms/step - loss: 0.8014 - accuracy:
0.6736 - val_loss: 0.8796 - val_accuracy: 0.6347
Epoch 69/100
471/471 [==============================] - ETA: 0s - loss: 0.8078 - accuracy: 0.6733
Epoch 69: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 35ms/step - loss: 0.8078 - accuracy:
0.6733 - val_loss: 0.7522 - val_accuracy: 0.7075
Epoch 70/100
471/471 [==============================] - ETA: 0s - loss: 0.7982 - accuracy: 0.6754
```

```
Epoch 70: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 34ms/step - loss: 0.7982 - accuracy:
0.6754 - val_loss: 0.8034 - val_accuracy: 0.6731
Epoch 71/100
469/471 [============================>.] - ETA: 0s - loss: 0.7997 - accuracy: 0.6788
Epoch 71: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 34ms/step - loss: 0.7999 - accuracy:
0.6787 - val_loss: 0.7993 - val_accuracy: 0.6659
Epoch 72/100
470/471 [============================>.] - ETA: 0s - loss: 0.7952 - accuracy: 0.6737
Epoch 72: val_accuracy did not improve from 0.70911
471/471 [==============================] - 16s 35ms/step - loss: 0.7948 - accuracy:
0.6739 - val_loss: 0.7346 - val_accuracy: 0.7065
```

In [67]:

```
plt.plot(history3.history['accuracy'])

plt.plot(history3.history['val_accuracy'])

plt.title('Model 3 - Convolutional Neural Network - Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc = 'upper left')

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



In [77]:

```
#test_images, test_labels = next(test_set)
#accuracy = model3.evaluate(test_images, test_labels, verbose = 2)
```

```
accuracy = model3.evaluate(test_set, verbose = 2)
```

```
4/4 - 0s - loss: 0.6360 - accuracy: 0.7578 - 133ms/epoch - 33ms/step
```

In [78]:

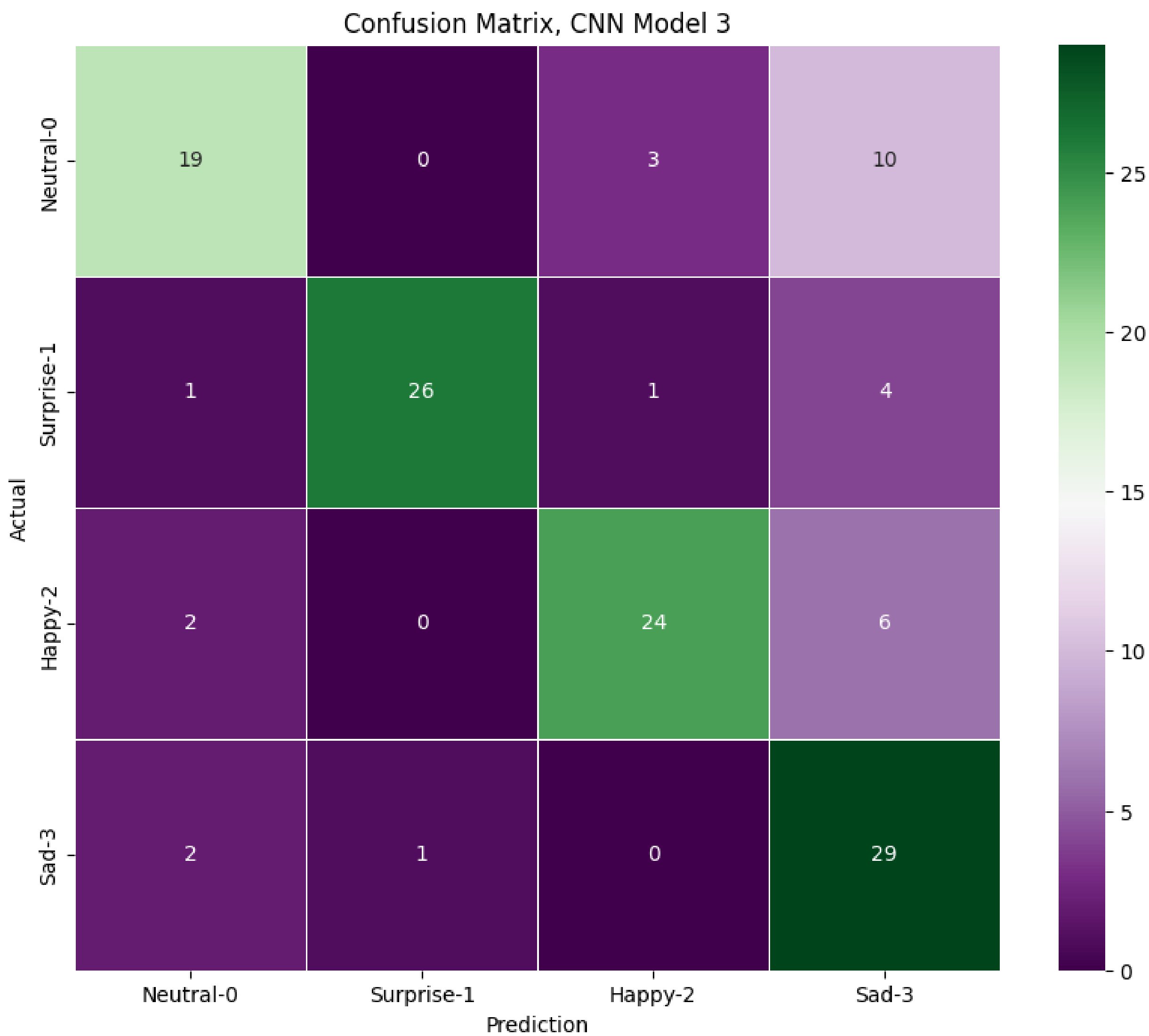
```
#code to get pairs of predicted vs. true for the confusion matrix
all_y_pred3 = []
all_y_true = []

for i in range(len(test_set)):
    x , y = test_set[i] #step through each test image/label
    y_pred3 = model3.predict(x) #run image though model, return predicted label

    #all_x.append(x)
    all_y_pred3.append(y_pred3) #add the predicted label to an ordered list
    all_y_true.append(y)      #add the true label to an ordered list

all_y_pred3 = np.concatenate(all_y_pred3, axis=0) #concatenate all preds
all_y_true = np.concatenate(all_y_true, axis=0) #concatenate all base vals


#test_images, test_labels = next(test_set)
#pred = model3.predict(test_images)
pred3 = np.argmax(all_y_pred3, axis=1) #get class indices
y_true = np.argmax(all_y_true, axis=1) #get class indices
```

```
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
```

In [79]:

```
# Printing the classification report
#importing function from sklearn
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(classification_report(y_true, pred3))

# Plotting the Confusion Matrix using confusion matrix() function
confusion_matrix3 = tf.math.confusion_matrix(y_true,pred3)
f, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(
    confusion_matrix3,
    annot=True,
    linewidths=.4,
    fmt="d",
    square=True,
    ax=ax,
    cmap="PRGn",
    xticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3'], #added labels for c
    yticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3'] #added labels for cl
)
plt.xlabel('Prediction',fontsize=10)
plt.ylabel('Actual',fontsize=10)
plt.title('Confusion Matrix, CNN Model 3', fontsize=12)
plt.show()
```

	precision	recall	f1-score	support
0	0.79	0.59	0.68	32
1	0.96	0.81	0.88	32
2	0.86	0.75	0.80	32
3	0.59	0.91	0.72	32
accuracy			0.77	128
macro avg	0.80	0.77	0.77	128
weighted avg	0.80	0.77	0.77	128



At first glance, this model - Model Three (3) - appears improve on Model Two (2). It certainly seems to take a turn for the better in making large strides towards classifying the “sad” class correctly, however, overall accuracy does not improve from Model Two (2).

This inherently means that while ability to classify the sad class correctly increases, the useability of the model is compromised because the *integrity of predictions for all other classes decreases* .

A Six Convolutional Block Model

(I said that I would try this in the milestone phase.)

In [106…
```
# Clearing backend

backend.clear_session()

#set random seed
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

In [107…
```
#hyperparameter tuner
#https://keras.io/guides/keras_tuner/getting_started/

#add leaky-relu parameter alpha=0.1 to the list of activation functions that are able
#from tensorflow.keras.utils import get_custom_objects
#get_custom_objects().update({'leaky-relu': Activation(LeakyReLU(alpha=0.1))})

#define drop rate
#drop=0.25

#define possible activation parameters for tuning
activation1 = "leaky-relu"
activation2 = "ReLU"
activation3 = "selu"

#define loss function for compiling the model
loss_function = 'categorical_crossentropy'

#tuning function
def build_model4(hp):

  drop=hp.Float("drop", min_value=0, max_value=0.5, step=0.05) #tuning drop rate
  activation=hp.Choice("activation", [activation1, activation2, activation3]) #tune th
  learning_rate = hp.Float("learning_rate", min_value=1e-4, max_value=1e-2, sampling='

# initialize model 2
  model4 = Sequential()

# First Convolutional Block
  units1=hp.Int("units1", min_value=32, max_value=512, step=32) #tune units to use for
  model4.add(Conv2D(units1, kernel_size=(3, 3), input_shape = (48, 48, 1), padding = '

# Second Convolutional Block
  units2=hp.Int("units2", min_value=32, max_value=512, step=32) #tune units to use for
  model4.add(Conv2D(units2, kernel_size=(3, 3), padding = 'same', activation=activatic
# Max Pooling
  model4.add(MaxPooling2D(2, 2))
# Add a dropout layer
  model4.add(Dropout(drop))
# Add a BatchNormalization layer
  model4.add(BatchNormalization())

# Third Convolutional Block
  units3=hp.Int("units3", min_value=32, max_value=512, step=32) #tune units to use for
  model4.add(Conv2D(units3, kernel_size=(2, 2), padding = 'same', activation=activatic
# Max Pooling
  model4.add(MaxPooling2D(2, 2))
```

```
# Add a dropout layer
  model4.add(Dropout(drop))
# Add a BatchNormalization layer
  model4.add(BatchNormalization())

# Fourth Convolutional Block
  units4=hp.Int("units4", min_value=32, max_value=512, step=32) #tune units to use for
  model4.add(Conv2D(units4, kernel_size=(3, 3), padding = 'same', activation=activatic
#Max Pooling
  model4.add(MaxPooling2D(2, 2))
#Add a dropout layer
  model4.add(Dropout(drop))
#Add a BatchNormalization layer
  model4.add(BatchNormalization())

  # Fifth Convolutional Block
  units5=hp.Int("units5", min_value=32, max_value=512, step=32) #tune units to use for
  model4.add(Conv2D(units5, kernel_size=(3, 3), padding = 'same', activation=activatic
#Max Pooling
  model4.add(MaxPooling2D(2, 2))
#Add a dropout layer
  model4.add(Dropout(drop))
#Add a BatchNormalization layer
  model4.add(BatchNormalization())

    # Sixth Convolutional Block
  units6=hp.Int("units6", min_value=32, max_value=512, step=32) #tune units to use for
  model4.add(Conv2D(units5, kernel_size=(3, 3), padding = 'same', activation=activatic
#Max Pooling
  model4.add(MaxPooling2D(1, 1))
#Add a dropout layer
  model4.add(Dropout(drop))
#Add a BatchNormalization layer
  model4.add(BatchNormalization())

#flatten
  model4.add(Flatten())

# First fully Connected Block
#default activation is relu
  unitsfc1=hp.Int("unitsfc1", min_value=32, max_value=512, step=32) #tune units to use
  model4.add(Dense(unitsfc1))
#Add a dropout layer
  model4.add(Dropout(drop))


# Second fully Connected Block
  unitsfc2=hp.Int("unitsfc2", min_value=32, max_value=512, step=32) #tune units to use
  model4.add(Dense(unitsfc2))
#Add a dropout layer
  model4.add(Dropout(drop))

  # Third fully Connected Block
  unitsfc3=hp.Int("unitsfc3", min_value=32, max_value=512, step=32) #tune units to use
  model4.add(Dense(unitsfc3))
#Add a dropout layer
#  model4.add(Dropout(drop))


# Classifier
```

```
  model4.add(Dense(4, activation = 'softmax'))

# Compiling the model
  model4.compile(loss = loss_function, optimizer=Adam(learning_rate=learning_rate) , m

 # model4.summary()
  return model4

build_model4(keras_tuner.HyperParameters())
```

Out[107]: <keras.src.engine.sequential.Sequential at 0x7bb7d8f08b50>

In [108…

```
# Set the path to the folder where you want to save the tuner output
tuner_path4 = "/content/drive/MyDrive/Tuners/"

tuner4 = keras_tuner.RandomSearch(
    hypermodel=build_model4,
    objective="val_accuracy",
    max_trials=3,
    executions_per_trial=3, #changed number of executions per trial from 2 to 3
    overwrite=True,
    directory=tuner_path4,
    project_name="Facial_Emotion_Milestone",
)
tuner4.search_space_summary()
```

```
Search space summary
Default search space size: 12
drop (Float)
{'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.5, 'step': 0.05,
'sampling': 'linear'}
activation (Choice)
{'default': 'leaky-relu', 'conditions': [], 'values': ['leaky-relu', 'ReLU', 'selu'],
'ordered': False}
learning_rate (Float)
{'default': 0.0001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.01, 'step':
None, 'sampling': 'log'}
units1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units3 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units4 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units5 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
units6 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
unitsfc3 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 's
ampling': 'linear'}
```

In [109…

```
tuner4.search(train_set, epochs=5, validation_data=validation_set) #increase number of
```

```
Trial 3 Complete [00h 04m 15s]
val_accuracy: 0.3259572237730026

Best val_accuracy So Far: 0.36779990792274475
Total elapsed time: 00h 12m 49s
```

In [110…

```
# Get the top model.
model4 = tuner4.get_best_models(num_models=1)
best_model4 = model4[0]
best_model4.build(input_shape=(48,48,1))
best_model4.summary()
```

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

 conv2d_1 (Conv2D)           (None, 48, 48, 352)       101728

 max_pooling2d (MaxPooling2  (None, 24, 24, 352)       0
 D)

 dropout (Dropout)           (None, 24, 24, 352)       0

 batch_normalization (Batch  (None, 24, 24, 352)       1408
 Normalization)

 conv2d_2 (Conv2D)           (None, 24, 24, 160)       225440

 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 160)       0
 g2D)

 dropout_1 (Dropout)         (None, 12, 12, 160)       0

 batch_normalization_1 (Bat  (None, 12, 12, 160)       640
 chNormalization)

 conv2d_3 (Conv2D)           (None, 12, 12, 64)        92224

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 dropout_2 (Dropout)         (None, 6, 6, 64)          0

 batch_normalization_2 (Bat  (None, 6, 6, 64)          256
 chNormalization)

 conv2d_4 (Conv2D)           (None, 6, 6, 512)         295424

 max_pooling2d_3 (MaxPoolin  (None, 3, 3, 512)         0
 g2D)

 dropout_3 (Dropout)         (None, 3, 3, 512)         0

 batch_normalization_3 (Bat  (None, 3, 3, 512)         2048
 chNormalization)

 conv2d_5 (Conv2D)           (None, 3, 3, 512)         2359808

 max_pooling2d_4 (MaxPoolin  (None, 3, 3, 512)         0
 g2D)

 dropout_4 (Dropout)         (None, 3, 3, 512)         0

 batch_normalization_4 (Bat  (None, 3, 3, 512)         2048
 chNormalization)

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 128)               589952
```

```
 dropout_5 (Dropout)         (None, 128)              0

 dense_1 (Dense)             (None, 64)               8256

 dropout_6 (Dropout)         (None, 64)               0

 dense_2 (Dense)             (None, 352)              22880

 dense_3 (Dense)             (None, 4)                1412

=================================================================
Total params: 3703844 (14.13 MB)
Trainable params: 3700644 (14.12 MB)
Non-trainable params: 3200 (12.50 KB)
_________________________________________________________________
```

In [111…

```
tuner4.results_summary()
```

```
Results summary
Results in /content/drive/MyDrive/Tuners/Facial_Emotion_Milestone
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 1 summary
Hyperparameters:
drop: 0.25
activation: ReLU
learning_rate: 0.00010224006994793711
units1: 32
units2: 352
units3: 160
units4: 64
units5: 512
units6: 64
unitsfc1: 128
unitsfc2: 64
unitsfc3: 352
Score: 0.36779990792274475

Trial 0 summary
Hyperparameters:
drop: 0.25
activation: ReLU
learning_rate: 0.0007614563868926456
units1: 96
units2: 96
units3: 352
units4: 352
units5: 192
units6: 480
unitsfc1: 384
unitsfc2: 352
unitsfc3: 32
Score: 0.3482867379983266

Trial 2 summary
Hyperparameters:
drop: 0.45
activation: ReLU
learning_rate: 0.0001702117816467284
units1: 128
units2: 64
units3: 224
units4: 32
units5: 128
units6: 480
unitsfc1: 224
unitsfc2: 384
unitsfc3: 224
Score: 0.3259572237730026
```

Compiling and Training the Model

In [112…

```
#drop=0.2

#COPY AND PASTE BEST MODEL FROM ABOVE, edit activation1 based on best params
#setting parameters
```

```
activation1 = "ReLU"
drop= 0.25
#activation: ReLU
learning_rate= 0.00010224   #006994793711
units1= 32
units2= 352
units3= 160
units4= 64
units5= 512
units6= 64
unitsfc1= 128
unitsfc2= 64
unitsfc3= 352
#Score: 0.36779990792274475


optimizer = Adam(learning_rate =learning_rate)


#initialize model
model4 = Sequential()

# First Convolutional layer with 32 filters and the kernel size of 3x3. Use the 'same'
model4.add(Conv2D(units1, kernel_size=(3, 3), input_shape = (48, 48, 1), padding = 'sa
#Max Pooling
model4.add(MaxPooling2D(2, 2))

# Second Convolutional Block
model4.add(Conv2D(units2, kernel_size=(2, 2), padding = 'same', activation=activation1
#Max Pooling
model4.add(MaxPooling2D(2, 2))
#Add a dropout layer
model4.add(Dropout(drop))
#Add a BatchNormalization layer
model4.add(BatchNormalization())

# Third Convolutional Block
model4.add(Conv2D(units3, kernel_size=(2, 2), padding = 'same', activation=activation1
#Max Pooling
model4.add(MaxPooling2D(2, 2))
#Add a dropout layer
model4.add(Dropout(drop))
#Add a BatchNormalization layer
model4.add(BatchNormalization())

# Fourth Convolutional Block
model4.add(Conv2D(units4, kernel_size=(2, 2), padding = 'same', activation=activation1
#Max Pooling
model4.add(MaxPooling2D(2, 2))
#Add a dropout layer
model4.add(Dropout(drop))
#Add a BatchNormalization layer
model4.add(BatchNormalization())

# Fifth Convolutional Block, default activation relu
model4.add(Conv2D(units5, kernel_size=(2, 2), padding = 'same')) #256
#Max Pooling
model4.add(MaxPooling2D(2, 2))
#Add a dropout layer
model4.add(Dropout(drop))
```

```python
#Add a BatchNormalization layer
model4.add(BatchNormalization())

# SIXTH Convolutional Block, default activation relu ADDING BECAUSE I SAID I WOULD
model4.add(Conv2D(units6, kernel_size=(2, 2), padding = 'same')) #256
#Max Pooling
model4.add(MaxPooling2D(1, 1))
#Add a dropout layer
model4.add(Dropout(drop))
#Add a BatchNormalization layer
model4.add(BatchNormalization())

#Flatten for dense layers
model4.add(Flatten())

# First fully Connected Block
#default activation is relu
model4.add(Dense(unitsfc1)) #512
#Add a dropout layer
model4.add(Dropout(drop))

# Second fully Connected Block
model4.add(Dense(unitsfc2)) #256
#Add a dropout layer
model4.add(Dropout(drop))

# Third fully Connected Block
model4.add(Dense(unitsfc3)) #128

#Add a dropout layer
#model4.add(Dropout(drop))


# Classifier Block
model4.add(Dense(4, activation = 'softmax'))
loss_function = 'categorical_crossentropy'

model4.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 48, 48, 32)        320       
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 24, 24, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_7 (Conv2D)           (None, 24, 24, 352)       45408     
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 12, 12, 352)       0         
 g2D)                                                            
                                                                 
 dropout_7 (Dropout)         (None, 12, 12, 352)       0         
                                                                 
 batch_normalization_5 (Bat  (None, 12, 12, 352)       1408      
 chNormalization)                                                
                                                                 
 conv2d_8 (Conv2D)           (None, 12, 12, 160)       225440    
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 6, 6, 160)         0         
 g2D)                                                            
                                                                 
 dropout_8 (Dropout)         (None, 6, 6, 160)         0         
                                                                 
 batch_normalization_6 (Bat  (None, 6, 6, 160)         640       
 chNormalization)                                                
                                                                 
 conv2d_9 (Conv2D)           (None, 6, 6, 64)          41024     
                                                                 
 max_pooling2d_8 (MaxPoolin  (None, 3, 3, 64)          0         
 g2D)                                                            
                                                                 
 dropout_9 (Dropout)         (None, 3, 3, 64)          0         
                                                                 
 batch_normalization_7 (Bat  (None, 3, 3, 64)          256       
 chNormalization)                                                
                                                                 
 conv2d_10 (Conv2D)          (None, 3, 3, 512)         131584    
                                                                 
 max_pooling2d_9 (MaxPoolin  (None, 1, 1, 512)         0         
 g2D)                                                            
                                                                 
 dropout_10 (Dropout)        (None, 1, 1, 512)         0         
                                                                 
 batch_normalization_8 (Bat  (None, 1, 1, 512)         2048      
 chNormalization)                                                
                                                                 
 conv2d_11 (Conv2D)          (None, 1, 1, 64)          131136    
                                                                 
 max_pooling2d_10 (MaxPooli  (None, 1, 1, 64)          0         
 ng2D)                                                           
                                                                 
 dropout_11 (Dropout)        (None, 1, 1, 64)          0         
                                                                 
 batch_normalization_9 (Bat  (None, 1, 1, 64)          256       
 chNormalization)                                                
                                                                 
 flatten_1 (Flatten)         (None, 64)                0         
```

```
 dense_4 (Dense)             (None, 128)               8320

 dropout_12 (Dropout)        (None, 128)               0

 dense_5 (Dense)             (None, 64)                8256

 dropout_13 (Dropout)        (None, 64)                0

 dense_6 (Dense)             (None, 352)               22880

 dense_7 (Dense)             (None, 4)                 1412

=================================================================
Total params: 620388 (2.37 MB)
Trainable params: 618084 (2.36 MB)
Non-trainable params: 2304 (9.00 KB)
_________________________________________________________________
```

In [113…

```
# Compiling the model
model4.compile(loss = loss_function, optimizer = optimizer , metrics = ['accuracy'])
```

In [114…

```
# setup checkpoint for early stopping
# Set the path to the folder where you want to save the checkpoint
checkpoint_path4 = "/content/drive/MyDrive/Colab_Checkpoints/checkpoint4.ckpt"

# Save the model's weights and optimizer state
model4.save_weights(checkpoint_path4)



early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.0005, patience=10, mode

#changed checkpoint parameters to save the model that gives the optimal val_accuracy
checkpoint4 = tf.keras.callbacks.ModelCheckpoint(checkpoint_path4,
                    monitor="val_accuracy", mode="max",
                    save_best_only=True, verbose=1)
```

In [115…

```
# Fitting the model
history4 = model4.fit(
    train_set,
    validation_data = validation_set,
    callbacks=[early_stopping, checkpoint4],
    epochs = 100)
```

```
Epoch 1/100
469/471 [============================>.] - ETA: 0s - loss: 1.4494 - accuracy: 0.2669
Epoch 1: val_accuracy improved from -inf to 0.36693, saving model to /content/drive/M
yDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 23s 40ms/step - loss: 1.4495 - accuracy:
0.2668 - val_loss: 1.3479 - val_accuracy: 0.3669
Epoch 2/100
471/471 [==============================] - ETA: 0s - loss: 1.3801 - accuracy: 0.2803
Epoch 2: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.3801 - accuracy:
0.2803 - val_loss: 1.3665 - val_accuracy: 0.2943
Epoch 3/100
470/471 [============================>.] - ETA: 0s - loss: 1.3759 - accuracy: 0.2804
Epoch 3: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 33ms/step - loss: 1.3759 - accuracy:
0.2803 - val_loss: 1.3650 - val_accuracy: 0.3082
Epoch 4/100
470/471 [============================>.] - ETA: 0s - loss: 1.3768 - accuracy: 0.2792
Epoch 4: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.3768 - accuracy:
0.2793 - val_loss: 1.3695 - val_accuracy: 0.2985
Epoch 5/100
470/471 [============================>.] - ETA: 0s - loss: 1.3724 - accuracy: 0.2850
Epoch 5: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 33ms/step - loss: 1.3723 - accuracy:
0.2853 - val_loss: 1.3669 - val_accuracy: 0.3120
Epoch 6/100
470/471 [============================>.] - ETA: 0s - loss: 1.3686 - accuracy: 0.2927
Epoch 6: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.3686 - accuracy:
0.2926 - val_loss: 1.3588 - val_accuracy: 0.3317
Epoch 7/100
469/471 [============================>.] - ETA: 0s - loss: 1.3660 - accuracy: 0.2928
Epoch 7: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 33ms/step - loss: 1.3660 - accuracy:
0.2927 - val_loss: 1.5116 - val_accuracy: 0.1957
Epoch 8/100
470/471 [============================>.] - ETA: 0s - loss: 1.3571 - accuracy: 0.3014
Epoch 8: val_accuracy did not improve from 0.36693
471/471 [==============================] - 16s 34ms/step - loss: 1.3571 - accuracy:
0.3015 - val_loss: 1.5097 - val_accuracy: 0.1915
Epoch 9/100
471/471 [==============================] - ETA: 0s - loss: 1.3518 - accuracy: 0.3135
Epoch 9: val_accuracy improved from 0.36693 to 0.38041, saving model to /content/driv
e/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.3518 - accuracy:
0.3135 - val_loss: 1.3282 - val_accuracy: 0.3804
Epoch 10/100
470/471 [============================>.] - ETA: 0s - loss: 1.3436 - accuracy: 0.3241
Epoch 10: val_accuracy did not improve from 0.38041
471/471 [==============================] - 16s 34ms/step - loss: 1.3436 - accuracy:
0.3242 - val_loss: 1.3501 - val_accuracy: 0.3213
Epoch 11/100
470/471 [============================>.] - ETA: 0s - loss: 1.3323 - accuracy: 0.3283
Epoch 11: val_accuracy did not improve from 0.38041
471/471 [==============================] - 16s 34ms/step - loss: 1.3323 - accuracy:
0.3286 - val_loss: 1.3073 - val_accuracy: 0.3422
Epoch 12/100
471/471 [==============================] - ETA: 0s - loss: 1.3234 - accuracy: 0.3415
Epoch 12: val_accuracy did not improve from 0.38041
```

```
471/471 [==============================] - 16s 33ms/step - loss: 1.3234 - accuracy:
0.3415 - val_loss: 1.4121 - val_accuracy: 0.2774
Epoch 13/100
470/471 [============================>.] - ETA: 0s - loss: 1.3119 - accuracy: 0.3528
Epoch 13: val_accuracy improved from 0.38041 to 0.40093, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 1.3118 - accuracy:
0.3529 - val_loss: 1.2644 - val_accuracy: 0.4009
Epoch 14/100
470/471 [============================>.] - ETA: 0s - loss: 1.2961 - accuracy: 0.3649
Epoch 14: val_accuracy improved from 0.40093 to 0.41098, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.2959 - accuracy:
0.3649 - val_loss: 1.2504 - val_accuracy: 0.4110
Epoch 15/100
471/471 [==============================] - ETA: 0s - loss: 1.2848 - accuracy: 0.3731
Epoch 15: val_accuracy did not improve from 0.41098
471/471 [==============================] - 16s 34ms/step - loss: 1.2848 - accuracy:
0.3731 - val_loss: 1.4701 - val_accuracy: 0.3084
Epoch 16/100
471/471 [==============================] - ETA: 0s - loss: 1.2736 - accuracy: 0.3824
Epoch 16: val_accuracy did not improve from 0.41098
471/471 [==============================] - 16s 33ms/step - loss: 1.2736 - accuracy:
0.3824 - val_loss: 1.2984 - val_accuracy: 0.3726
Epoch 17/100
471/471 [==============================] - ETA: 0s - loss: 1.2602 - accuracy: 0.3905
Epoch 17: val_accuracy did not improve from 0.41098
471/471 [==============================] - 17s 35ms/step - loss: 1.2602 - accuracy:
0.3905 - val_loss: 1.2721 - val_accuracy: 0.3959
Epoch 18/100
470/471 [============================>.] - ETA: 0s - loss: 1.2432 - accuracy: 0.4079
Epoch 18: val_accuracy improved from 0.41098 to 0.44780, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 41ms/step - loss: 1.2436 - accuracy:
0.4076 - val_loss: 1.1942 - val_accuracy: 0.4478
Epoch 19/100
470/471 [============================>.] - ETA: 0s - loss: 1.2349 - accuracy: 0.4178
Epoch 19: val_accuracy improved from 0.44780 to 0.45202, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.2352 - accuracy:
0.4176 - val_loss: 1.1778 - val_accuracy: 0.4520
Epoch 20/100
471/471 [==============================] - ETA: 0s - loss: 1.2142 - accuracy: 0.4316
Epoch 20: val_accuracy improved from 0.45202 to 0.48079, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.2142 - accuracy:
0.4316 - val_loss: 1.1549 - val_accuracy: 0.4808
Epoch 21/100
469/471 [============================>.] - ETA: 0s - loss: 1.2081 - accuracy: 0.4355
Epoch 21: val_accuracy did not improve from 0.48079
471/471 [==============================] - 16s 33ms/step - loss: 1.2085 - accuracy:
0.4353 - val_loss: 1.2283 - val_accuracy: 0.4367
Epoch 22/100
471/471 [==============================] - ETA: 0s - loss: 1.1907 - accuracy: 0.4470
Epoch 22: val_accuracy did not improve from 0.48079
471/471 [==============================] - 16s 34ms/step - loss: 1.1907 - accuracy:
0.4470 - val_loss: 1.1590 - val_accuracy: 0.4371
Epoch 23/100
470/471 [============================>.] - ETA: 0s - loss: 1.1761 - accuracy: 0.4558
Epoch 23: val_accuracy did not improve from 0.48079
```

```
471/471 [==============================] - 16s 33ms/step - loss: 1.1764 - accuracy: 
0.4555 - val_loss: 1.1585 - val_accuracy: 0.4577
Epoch 24/100
470/471 [============================>.] - ETA: 0s - loss: 1.1630 - accuracy: 0.4607
Epoch 24: val_accuracy improved from 0.48079 to 0.49628, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 41ms/step - loss: 1.1632 - accuracy: 
0.4605 - val_loss: 1.0932 - val_accuracy: 0.4963
Epoch 25/100
470/471 [============================>.] - ETA: 0s - loss: 1.1495 - accuracy: 0.4793
Epoch 25: val_accuracy improved from 0.49628 to 0.53732, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.1495 - accuracy: 
0.4791 - val_loss: 1.0806 - val_accuracy: 0.5373
Epoch 26/100
471/471 [==============================] - ETA: 0s - loss: 1.1393 - accuracy: 0.4856
Epoch 26: val_accuracy did not improve from 0.53732
471/471 [==============================] - 16s 34ms/step - loss: 1.1393 - accuracy: 
0.4856 - val_loss: 1.0592 - val_accuracy: 0.5335
Epoch 27/100
471/471 [==============================] - ETA: 0s - loss: 1.1295 - accuracy: 0.4924
Epoch 27: val_accuracy did not improve from 0.53732
471/471 [==============================] - 16s 33ms/step - loss: 1.1295 - accuracy: 
0.4924 - val_loss: 1.0941 - val_accuracy: 0.5110
Epoch 28/100
470/471 [============================>.] - ETA: 0s - loss: 1.1082 - accuracy: 0.5015
Epoch 28: val_accuracy improved from 0.53732 to 0.55281, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 41ms/step - loss: 1.1081 - accuracy: 
0.5014 - val_loss: 1.0352 - val_accuracy: 0.5528
Epoch 29/100
471/471 [==============================] - ETA: 0s - loss: 1.1030 - accuracy: 0.5158
Epoch 29: val_accuracy improved from 0.55281 to 0.56327, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 1.1030 - accuracy: 
0.5158 - val_loss: 1.0154 - val_accuracy: 0.5633
Epoch 30/100
469/471 [============================>.] - ETA: 0s - loss: 1.0836 - accuracy: 0.5238
Epoch 30: val_accuracy improved from 0.56327 to 0.56709, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.0832 - accuracy: 
0.5241 - val_loss: 0.9907 - val_accuracy: 0.5671
Epoch 31/100
469/471 [============================>.] - ETA: 0s - loss: 1.0787 - accuracy: 0.5273
Epoch 31: val_accuracy improved from 0.56709 to 0.57654, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.0789 - accuracy: 
0.5272 - val_loss: 1.0002 - val_accuracy: 0.5765
Epoch 32/100
470/471 [============================>.] - ETA: 0s - loss: 1.0675 - accuracy: 0.5321
Epoch 32: val_accuracy improved from 0.57654 to 0.58982, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.0673 - accuracy: 
0.5321 - val_loss: 0.9726 - val_accuracy: 0.5898
Epoch 33/100
470/471 [============================>.] - ETA: 0s - loss: 1.0639 - accuracy: 0.5367
Epoch 33: val_accuracy did not improve from 0.58982
471/471 [==============================] - 16s 33ms/step - loss: 1.0643 - accuracy: 
0.5364 - val_loss: 0.9888 - val_accuracy: 0.5844
Epoch 34/100
```

```
471/471 [==============================] - ETA: 0s - loss: 1.0457 - accuracy: 0.5492
Epoch 34: val_accuracy improved from 0.58982 to 0.59767, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.0457 - accuracy:
0.5492 - val_loss: 0.9647 - val_accuracy: 0.5977
Epoch 35/100
471/471 [==============================] - ETA: 0s - loss: 1.0305 - accuracy: 0.5551
Epoch 35: val_accuracy improved from 0.59767 to 0.61336, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 1.0305 - accuracy:
0.5551 - val_loss: 0.9261 - val_accuracy: 0.6134
Epoch 36/100
471/471 [==============================] - ETA: 0s - loss: 1.0307 - accuracy: 0.5534
Epoch 36: val_accuracy did not improve from 0.61336
471/471 [==============================] - 16s 33ms/step - loss: 1.0307 - accuracy:
0.5534 - val_loss: 0.9494 - val_accuracy: 0.5987
Epoch 37/100
471/471 [==============================] - ETA: 0s - loss: 1.0267 - accuracy: 0.5583
Epoch 37: val_accuracy did not improve from 0.61336
471/471 [==============================] - 16s 34ms/step - loss: 1.0267 - accuracy:
0.5583 - val_loss: 0.9318 - val_accuracy: 0.6085
Epoch 38/100
471/471 [==============================] - ETA: 0s - loss: 1.0112 - accuracy: 0.5646
Epoch 38: val_accuracy did not improve from 0.61336
471/471 [==============================] - 16s 34ms/step - loss: 1.0112 - accuracy:
0.5646 - val_loss: 0.9807 - val_accuracy: 0.5816
Epoch 39/100
470/471 [============================>.] - ETA: 0s - loss: 1.0072 - accuracy: 0.5698
Epoch 39: val_accuracy improved from 0.61336 to 0.61456, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.0070 - accuracy:
0.5699 - val_loss: 0.9308 - val_accuracy: 0.6146
Epoch 40/100
471/471 [==============================] - ETA: 0s - loss: 1.0095 - accuracy: 0.5668
Epoch 40: val_accuracy improved from 0.61456 to 0.63368, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 1.0095 - accuracy:
0.5668 - val_loss: 0.8944 - val_accuracy: 0.6337
Epoch 41/100
471/471 [==============================] - ETA: 0s - loss: 0.9920 - accuracy: 0.5751
Epoch 41: val_accuracy did not improve from 0.63368
471/471 [==============================] - 16s 33ms/step - loss: 0.9920 - accuracy:
0.5751 - val_loss: 0.8953 - val_accuracy: 0.6222
Epoch 42/100
470/471 [============================>.] - ETA: 0s - loss: 1.0039 - accuracy: 0.5677
Epoch 42: val_accuracy improved from 0.63368 to 0.63388, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 1.0038 - accuracy:
0.5676 - val_loss: 0.8899 - val_accuracy: 0.6339
Epoch 43/100
470/471 [============================>.] - ETA: 0s - loss: 0.9918 - accuracy: 0.5802
Epoch 43: val_accuracy did not improve from 0.63388
471/471 [==============================] - 16s 34ms/step - loss: 0.9918 - accuracy:
0.5805 - val_loss: 0.8937 - val_accuracy: 0.6297
Epoch 44/100
471/471 [==============================] - ETA: 0s - loss: 0.9899 - accuracy: 0.5817
Epoch 44: val_accuracy improved from 0.63388 to 0.63750, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.9899 - accuracy:
0.5817 - val_loss: 0.8649 - val_accuracy: 0.6375
```

```
Epoch 45/100
470/471 [============================>.] - ETA: 0s - loss: 0.9712 - accuracy: 0.5923
Epoch 45: val_accuracy improved from 0.63750 to 0.65218, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.9714 - accuracy:
0.5919 - val_loss: 0.8433 - val_accuracy: 0.6522
Epoch 46/100
471/471 [==============================] - ETA: 0s - loss: 0.9847 - accuracy: 0.5793
Epoch 46: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 34ms/step - loss: 0.9847 - accuracy:
0.5793 - val_loss: 0.8568 - val_accuracy: 0.6518
Epoch 47/100
469/471 [============================>.] - ETA: 0s - loss: 0.9631 - accuracy: 0.5921
Epoch 47: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 34ms/step - loss: 0.9631 - accuracy:
0.5921 - val_loss: 0.8609 - val_accuracy: 0.6453
Epoch 48/100
471/471 [==============================] - ETA: 0s - loss: 0.9616 - accuracy: 0.5918
Epoch 48: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 33ms/step - loss: 0.9616 - accuracy:
0.5918 - val_loss: 0.8568 - val_accuracy: 0.6484
Epoch 49/100
471/471 [==============================] - ETA: 0s - loss: 0.9558 - accuracy: 0.5973
Epoch 49: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 34ms/step - loss: 0.9558 - accuracy:
0.5973 - val_loss: 0.9125 - val_accuracy: 0.6152
Epoch 50/100
470/471 [============================>.] - ETA: 0s - loss: 0.9508 - accuracy: 0.6050
Epoch 50: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 33ms/step - loss: 0.9506 - accuracy:
0.6053 - val_loss: 0.8729 - val_accuracy: 0.6357
Epoch 51/100
470/471 [============================>.] - ETA: 0s - loss: 0.9446 - accuracy: 0.6040
Epoch 51: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 34ms/step - loss: 0.9446 - accuracy:
0.6043 - val_loss: 0.8639 - val_accuracy: 0.6425
Epoch 52/100
471/471 [==============================] - ETA: 0s - loss: 0.9505 - accuracy: 0.5987
Epoch 52: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 34ms/step - loss: 0.9505 - accuracy:
0.5987 - val_loss: 0.8556 - val_accuracy: 0.6480
Epoch 53/100
470/471 [============================>.] - ETA: 0s - loss: 0.9405 - accuracy: 0.6055
Epoch 53: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 35ms/step - loss: 0.9408 - accuracy:
0.6056 - val_loss: 0.8643 - val_accuracy: 0.6431
Epoch 54/100
470/471 [============================>.] - ETA: 0s - loss: 0.9437 - accuracy: 0.6051
Epoch 54: val_accuracy did not improve from 0.65218
471/471 [==============================] - 16s 33ms/step - loss: 0.9435 - accuracy:
0.6054 - val_loss: 0.8493 - val_accuracy: 0.6427
Epoch 55/100
470/471 [============================>.] - ETA: 0s - loss: 0.9316 - accuracy: 0.6103
Epoch 55: val_accuracy improved from 0.65218 to 0.65419, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.9317 - accuracy:
0.6101 - val_loss: 0.8192 - val_accuracy: 0.6542
Epoch 56/100
470/471 [============================>.] - ETA: 0s - loss: 0.9329 - accuracy: 0.6070
Epoch 56: val_accuracy did not improve from 0.65419
```

```
471/471 [==============================] - 16s 34ms/step - loss: 0.9330 - accuracy:
0.6070 - val_loss: 0.8511 - val_accuracy: 0.6459
Epoch 57/100
469/471 [============================>.] - ETA: 0s - loss: 0.9251 - accuracy: 0.6153
Epoch 57: val_accuracy improved from 0.65419 to 0.66465, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 18s 39ms/step - loss: 0.9244 - accuracy:
0.6156 - val_loss: 0.8115 - val_accuracy: 0.6647
Epoch 58/100
471/471 [==============================] - ETA: 0s - loss: 0.9265 - accuracy: 0.6121
Epoch 58: val_accuracy improved from 0.66465 to 0.66908, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 39ms/step - loss: 0.9265 - accuracy:
0.6121 - val_loss: 0.8204 - val_accuracy: 0.6691
Epoch 59/100
470/471 [============================>.] - ETA: 0s - loss: 0.9261 - accuracy: 0.6162
Epoch 59: val_accuracy did not improve from 0.66908
471/471 [==============================] - 16s 34ms/step - loss: 0.9260 - accuracy:
0.6160 - val_loss: 0.8179 - val_accuracy: 0.6632
Epoch 60/100
469/471 [============================>.] - ETA: 0s - loss: 0.9178 - accuracy: 0.6199
Epoch 60: val_accuracy improved from 0.66908 to 0.67371, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 41ms/step - loss: 0.9167 - accuracy:
0.6205 - val_loss: 0.7954 - val_accuracy: 0.6737
Epoch 61/100
471/471 [==============================] - ETA: 0s - loss: 0.9227 - accuracy: 0.6157
Epoch 61: val_accuracy did not improve from 0.67371
471/471 [==============================] - 16s 33ms/step - loss: 0.9227 - accuracy:
0.6157 - val_loss: 0.8193 - val_accuracy: 0.6645
Epoch 62/100
470/471 [============================>.] - ETA: 0s - loss: 0.9083 - accuracy: 0.6226
Epoch 62: val_accuracy did not improve from 0.67371
471/471 [==============================] - 16s 34ms/step - loss: 0.9084 - accuracy:
0.6226 - val_loss: 0.8197 - val_accuracy: 0.6550
Epoch 63/100
470/471 [============================>.] - ETA: 0s - loss: 0.9046 - accuracy: 0.6248
Epoch 63: val_accuracy did not improve from 0.67371
471/471 [==============================] - 16s 34ms/step - loss: 0.9044 - accuracy:
0.6251 - val_loss: 0.8247 - val_accuracy: 0.6616
Epoch 64/100
471/471 [==============================] - ETA: 0s - loss: 0.9075 - accuracy: 0.6245
Epoch 64: val_accuracy improved from 0.67371 to 0.67612, saving model to /content/dri
ve/MyDrive/Colab_Checkpoints/checkpoint4.ckpt
471/471 [==============================] - 19s 40ms/step - loss: 0.9075 - accuracy:
0.6245 - val_loss: 0.7968 - val_accuracy: 0.6761
Epoch 65/100
469/471 [============================>.] - ETA: 0s - loss: 0.9141 - accuracy: 0.6189
Epoch 65: val_accuracy did not improve from 0.67612
471/471 [==============================] - 16s 34ms/step - loss: 0.9137 - accuracy:
0.6192 - val_loss: 0.7990 - val_accuracy: 0.6727
Epoch 66/100
471/471 [==============================] - ETA: 0s - loss: 0.9010 - accuracy: 0.6238
Epoch 66: val_accuracy did not improve from 0.67612
471/471 [==============================] - 16s 34ms/step - loss: 0.9010 - accuracy:
0.6238 - val_loss: 0.8107 - val_accuracy: 0.6683
Epoch 67/100
471/471 [==============================] - ETA: 0s - loss: 0.9046 - accuracy: 0.6277
Epoch 67: val_accuracy did not improve from 0.67612
471/471 [==============================] - 17s 35ms/step - loss: 0.9046 - accuracy:
```

```
0.6277 - val_loss: 0.8131 - val_accuracy: 0.6713
Epoch 68/100
471/471 [==============================] - ETA: 0s - loss: 0.8977 - accuracy: 0.6236
Epoch 68: val_accuracy did not improve from 0.67612
471/471 [==============================] - 16s 34ms/step - loss: 0.8977 - accuracy:
0.6236 - val_loss: 0.8065 - val_accuracy: 0.6703
Epoch 69/100
470/471 [============================>.] - ETA: 0s - loss: 0.8968 - accuracy: 0.6272
Epoch 69: val_accuracy did not improve from 0.67612
471/471 [==============================] - 16s 34ms/step - loss: 0.8971 - accuracy:
0.6268 - val_loss: 0.8209 - val_accuracy: 0.6639
Epoch 70/100
471/471 [==============================] - ETA: 0s - loss: 0.8953 - accuracy: 0.6281
Epoch 70: val_accuracy did not improve from 0.67612
471/471 [==============================] - 16s 34ms/step - loss: 0.8953 - accuracy:
0.6281 - val_loss: 0.8466 - val_accuracy: 0.6490
```

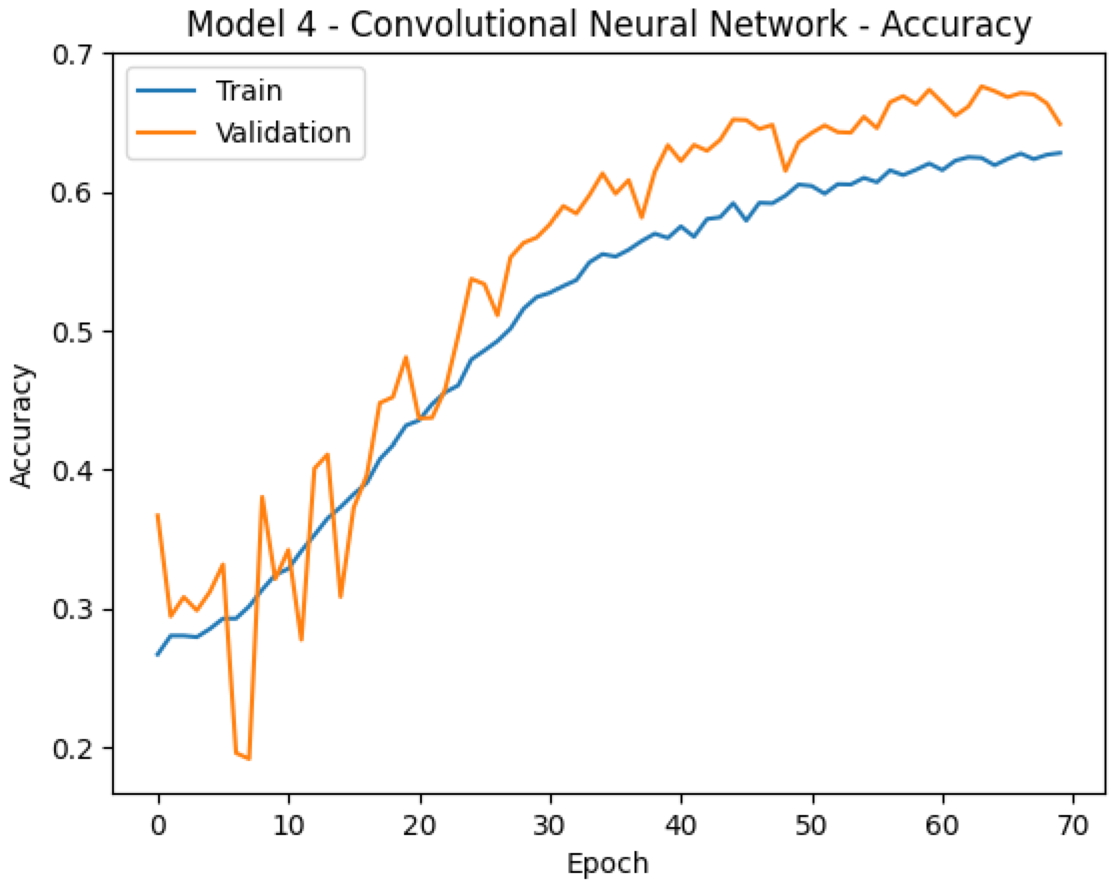
In [116…

```
#save model4
model4.save("facialemotionmodel4.h5py")
```

Evaluating the Model on Test Set

In [117…

```
plt.plot(history4.history['accuracy'])

plt.plot(history4.history['val_accuracy'])

plt.title('Model 4 - Convolutional Neural Network - Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc = 'upper left')

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



Evaluating the Model on the Test Set

In [121…

```
#test_images, test_labels = next(test_set)
#accuracy = model4.evaluate(test_images, test_labels, verbose = 2)
accuracy = model4.evaluate(test_set, verbose = 2)
```

```
4/4 - 0s - loss: 0.7840 - accuracy: 0.6953 - 139ms/epoch - 35ms/step
```

Observations and Insights:__

Plotting the Confusion Matrix for the chosen final model

In [124…

```
#code to get pairs of predicted vs. true for the confusion matrix
all_y_pred4 = []
all_y_true = []

for i in range(len(test_set)):
    x , y = test_set[i] #step through each test image/label
    y_pred4 = model4.predict(x) #run image though model, return predicted label

    #all_x.append(x)
    all_y_pred4.append(y_pred4) #add the predicted label to an ordered list
    all_y_true.append(y)      #add the true label to an ordered list

all_y_pred4 = np.concatenate(all_y_pred4, axis=0) #concatenate all preds
all_y_true = np.concatenate(all_y_true, axis=0) #concatenate all base vals


#test_images, test_labels = next(test_set)
```

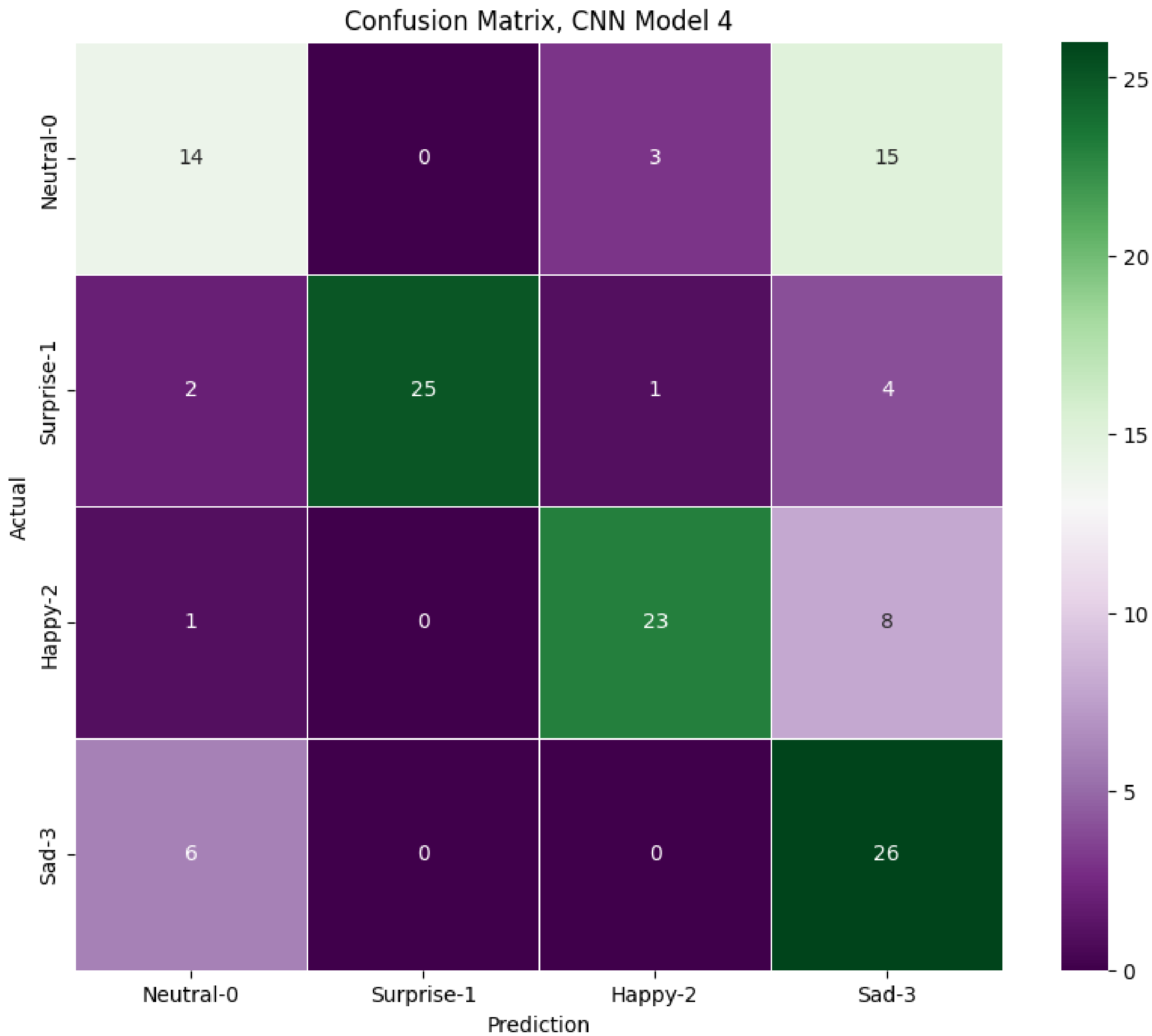
```python
#pred = model4.predict(test_images)
pred4 = np.argmax(all_y_pred4, axis=1) #get class indices
y_true = np.argmax(all_y_true, axis=1) #get class indices
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
```

In [125…

```python
# Printing the classification report
#importing function from sklearn
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(classification_report(y_true, pred4))

# Plotting the Confusion Matrix using confusion matrix() function
confusion_matrix4 = tf.math.confusion_matrix(y_true,pred4)
f, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(
    confusion_matrix4,
    annot=True,
    linewidths=.4,
    fmt="d",
    square=True,
    ax=ax,
    cmap="PRGn", # changed color mode to compliment presentation format
    xticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3'], #added labels for c
    yticklabels = ['Neutral-0', 'Surprise-1', 'Happy-2', 'Sad-3'] #added labels for cl
)
plt.xlabel('Prediction',fontsize=10)
plt.ylabel('Actual',fontsize=10)
plt.title('Confusion Matrix, CNN Model 4', fontsize=12)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.61      0.44      0.51        32
           1       1.00      0.78      0.88        32
           2       0.85      0.72      0.78        32
           3       0.49      0.81      0.61        32

    accuracy                           0.69       128
   macro avg       0.74      0.69      0.69       128
weighted avg       0.74      0.69      0.69       128
```



**Observations and Insights:

Model Four (4), again, does a better job of classifing sad faces, and recall of sad faces is very high, however, predictions are not precise, due to the persistent overlap with the neutral class.

This is encouraging and shows that artificial discernment of the classes is possible, though it will require more rigorous methods of pushing classification into appropriate buckets.

Of course, one method of achieving this would be to "instruct" the model. That is, by severely exaggerating the characteristics that *we think should define sad*, and to give clear markers of "sadness" so that "sadness" is easily identifiable almost in the same way that "surprised" is identifiable.

Again, clearly human emotions are not classifiable into discrete, neat buckets, and it necessary to attempt such an exercise with this in mind.

Overall model accuracy is okay, and accuracy for the "surprised" and "happy classes" are fair, but they are not as good as Model 2.

The progression of models clearly illustrates the consistent struggle to find clear distinguishing features that separate neutral from the other three classes.

classes:
neutral 0
surprise 1
happy 2
sad 3