

Application of Machine Learning Assignment 2

by Gowri Pamarthi

Submission date: 04-May-2023 12:59PM (UTC+0100)

Submission ID: 204692592

File name:

42643_Gowri_Pamarthi_Application_of_Machine_Learning_Assignment_2_498640_104026693.docx (1.48M)

Word count: 4422

Character count: 25829



Student Name: GOWRI PAMARTHI

Student Number: 2206771

Module Name:

Application of Machine Learning (ACCA7025)

Assignment Title:

Facial Expression Recognition by using CNN

Table of Contents

Table of Contents	2
List of Figure	3
List of Listing	3
1 Introduction	4
1.1 Overview	4
1.2 Dataset Examination or Description	4
1.3 Deep Neural Networks (DNN)	4
1.4 Convolutional Neural Networks (CNN)	5
1.4.1 Key points of CNN	5
1.4.2 Advantages of CNN	6
1.4.3 Disadvantages of CNN	6
1.5 CNN model architecture	6
2 Project implementation with Result and Analysis	9
3 Conclusion, Recommendation, and Future Scope	22
3.1 Conclusion	22
3.2 Recommendation	22
3.3 Future Scope	22
4 References	23
5 Appendices	24
6 Video link	33

List of Figure



Figure 1.1 Summary of the model architecture	8
Figure 2.1 Different types of expression name	9
Figure 2.2 shows the expression of class 0.	10
Figure 2.3 shows all the angry image which is present in the dataset.	10
Figure 2.4 show the image data of disgusting which is available in the dataset	11
Figure 2.5 display the entire fear image which is present in this data.	11
Figure 2.6 shows all the happy images	12
Figure 2.7 demonstrates all the neutral images	12
Figure 2.8 display all the sad image which is present in this dataset.	13
Figure 2.9 shows all the surprise pictures	14
Figure 2.10 display all the class for comparing one class to another class.	14
Figure 2.11 divide all the dataset into training data and testing data.	15
Figure 2.12 shows the summary of the model.	17
Figure 2.13 shows the first epochs	18
Figure 2.14 shows the last epochs.	19
Figure 2.15 evaluate the model	19
Figure 2.16 display the training and testing accuracy graphically.	20

List of Listing



Listing 2.1 Load all the important Libraries to execute the program.	9
Listing 2.2 Load the angry dataset.	10
Listing 2.3 Load disgusting dataset	10
Listing 2.4 Load fear image	11
Listing 2.5 loads the happy image	12
Listing 2.6 load neutral images	12
Listing 2.7 loads sad images	13
Listing 2.8 surprise image loading with the help of cv	13
Listing 2.9 Creating the model using CNN	16
Listing 2.10 compile the model with the help of Adam optimizers.....	17
Listing 2.11 create the plot for showing the train accuracy and test accuracy.....	20
Listing 2.12 save the model.	20

1 Introduction

1.1 Overview

In the advanced technological world, there are almost real-time problems that are trying to sort out using advanced and modern tools, and different researchers got succeeded also [1]. Normally a human can say by seeing a person that the person is happy or sad etc. which means the emotion behind the facial expression. But if a machine can do the same thing in a very accurate way, then it is amazing because without human attachment and humans may did mistake to identify emotions but machines cannot. Here the author tries to create a model which has the ability to find the different emotions behind the facial expression of humans. To execute this, aim the author takes the help of different Deep Learning (DL) tools such as Deep Neural Networks (DNN), Convolution Neural Networks (CNN), etc. Here the author tries to create the best suitable model with different activation functions, dense layers, and many more so that the author gets obtained an accurate result through the model performance [2]. As it tries to classify certain emotions of humans so that it comes under the classification problem of DL and here the author has to create a classification model.

1.2 Dataset Examination or Description

Here the author used different images of emotions of human expression because a human has only certain different emotions only such as anger, disgust, fear, happiness, neutral, sad, and surprise. Here 3995 angry images, 436 disgusting images, 4097 fear images, 7215 happy images, 4965 neutral images, 4830 sad images, and 3171 images of surprise images are available in the training dataset. All the images are stored in a list format with their expressed emotions to be passed in the CNN model to train the model so that model can learn all the hidden patterns to understand the emotions behind the expression.

1.3 Deep Neural Networks (DNN)

DNN is a combination of ANN with multiple hidden layers and these layers are situated between the input and output layers [3]. The main objective of this kind of neural network is to get complex data as input and perform complex calculations in a higher accurate way to solve any real-time problem statement. This technique is mostly used in any supervised learning techniques as well as reinforcement learning techniques also because it has the capability to deal with any large amount of data with higher dimensions, even if it is considered that this network works better only when the data is available in a bulk amount. It is also considered that if the data pattern is complex then a higher number of hidden layers gives the

higher or greater or best optimal output [4]. Inside the layer there are multiple mathematical calculations are going on because of the data weightage. In every layer, the data point gets some amount of weightage and it is dependent on different probabilistic calculations. Inside the network, it looks like a black hole because the user doesn't know what weightage values are assigned to which data means the machine decided himself based on some probability values [5]. So, there is nothing in the user's hand, the user just passes input data to the model and the model himself do the remaining prediction work and weightage distribution. Without human intervention, deep learning uses feature transformation and pattern recognition to classify images into appropriate categories. Depending on the type, DNN comes in a variety of data. DNNs include a variety of techniques or types. such as convolutional neural networks (CNN), recurrent neural networks (RNN), and long short-term memory (LSTM). Although there are different techniques available in DNN the author implemented only CNN for this proposed work-study [6].

1.4 Convolutional Neural Networks (CNN)

Using one of the well-known artificial neural networks, CNN, picture recognition, and image processing are developed for pixel data processing in this method. This style of image analysis is the most common. Since its inception, CNN designs have advanced quickly, and most recently they have led to successes that were once believed to require human execution or involvement. Depending on the work at hand and the accompanying constraints, a variety of architectures are currently available. In the 1980s, Yann LeCun, a postdoctoral researcher in computer science, developed the first convolution neural network [7]. LeNet, which was a prototype of ConvNets and was called after LeCun, was able to read handwritten digits. This type of neural network requires substantially less pre-processing as compared to other classification techniques. ConvNets enable learning of certain characteristics/filters, which are hand-engineered in simple methods with sufficient training. The descriptive and generating task for using deep learning techniques where machine vision is used to including video and image recognition, natural language processing, and recommender systems.

1.4.1 Key points of CNN

It is a class of DL techniques that have taken control of the numerous CV tasks and is luring users from a variety of fields, especially radiology.

- Through the use of a backpropagation algorithm, it consists of multiple building blocks, including convolution layers, pooling layers and fully connected layers, and is intended rapidly and adaptively learn a spatial hierarchy of information.
- It has the potential to enhance radiologists' efficiency and, ultimately, the treatment of patients.



1.4.2 Advantages of CNN

Image processing: CNN can entail using a selection of filters to extract characteristics of a picture that are pertinent to the task at hand. CNNs are quicker and more effective than other types of algorithms because they may reduce the amount of information that needs to be analyzed.

High accuracy: By examining enormous databases of photos, it can train itself to recognize intricate patterns.

Transfer learning: Additionally, because CNNs support transfer learning, they can be taught to execute one task and then applied to another one with little to no extra instruction.

Automated feature extraction: Last but not least, since CNNs automate the process of extracting features, they may learn to detect patterns in images without the need for manual feature engineering [8].

1.4.3 Disadvantages of CNN

Computational requirements: Most CNNs include a lot of layers and parameters, thus training and using them takes a lot of computing power and memory.

Difficulty with small datasets: The CNN might overfit, or become overly specialized to the training dataset and perform badly on fresh data if the dataset is too small.

Limited ability to generalize: At last, CNNs can only a limited extent generalize to novel circumstances [9]. This implies that they can struggle to do well on pictures that are very dissimilar from the training sample.

1.5 CNN model architecture

Dataset: Here in this proposed project is face emotion detection data. Here are 7 types of classes. such as anger, disgust, fear, happy, neutral, sadness and surprise all in image format.

Layers: There are four layers of neurons in this structure. The author in this case employed two convolution networks with kernel sizes of 3. It indicates that the output for the following layer will be determined using the image matrix and a 3x3 filter.

Number of Neurons: The first layer has 64 nodes or neurons, the second layer has half as many, and the third layer has four nodes or neurons because there are four classes altogether. This hyperparameter has been set differently in each layer. The likelihood of each of the four classes in the output should be used to determine which class should be the projected output.

Activation function: A rectified linear unit activation function fixes the issue of vanishing gradients and gives a deep-learning model of the non-linearity property. 0.0 is returned in place of the positive value when the value is less than (negative values) or equal to zero.

Optimizers: It is used to cut loss down to the bare minimum. It alters the learning rate and weights to obtain the least amount of loss. To determine the emotion, the Adam (Adaptive moment estimate) optimizer was used.

Learning rate: It is an hyperparameter with a default value of 0.001 for Adam optimizers. Let the learning rate determine the step size and to obtain the least loss. But here the author uses the learning rate as 0.0005.

Epochs: The frequency at which a collection of datasets is delivered throughout the entire network is controlled by a hyperparameter. The epoch has been set to 15 for this CNN.

Dropout: Here the author uses dropout because it helps to reduce the chances of overfitting of the model.

Batch Normalization: the purpose of using this technique is to give the freedom to learn all the patterns independently and more accurately.

Maxpool2D: This technique helps a lot to train the model when dealing with image data because training with the help of this techniques model reduces the dimensionality of the model.

```
model.summary() # showing the summary of the designed model
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	320
conv2d_1 (Conv2D)	(None, 48, 48, 64)	18496
batch_normalization (Batch Normalization)	(None, 48, 48, 64)	256
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0
dropout (Dropout)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	73856
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295168
batch_normalization_1 (Batch Normalization)	(None, 24, 24, 256)	1024
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 256)	0
dropout_1 (Dropout)	(None, 12, 12, 256)	0
conv2d_4 (Conv2D)	(None, 12, 12, 256)	590080
batch_normalization_2 (Batch Normalization)	(None, 12, 12, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
dropout_2 (Dropout)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 100)	921700
dense_1 (Dense)	(None, 50)	5050
dense_2 (Dense)	(None, 225)	11475
batch_normalization_3 (Batch Normalization)	(None, 225)	900
dropout_3 (Dropout)	(None, 225)	0
dense_3 (Dense)	(None, 7)	1582
Total params: 1,920,931		
Trainable params: 1,919,329		
Non-trainable params: 1,602		

Figure 1.1 Summary of the model architecture

2 Project implementation with Result and Analysis

In this section, the author discusses all the steps which are used in this project to execute the project according to the proposed project the author uses CNN techniques to predict the model and also the author processes the image data with the help of DL techniques. Because it is image-based data so before building the model it must be important to process all the images and convert them into matrix format.

Now the author describes all the steps one by one below:

Everyone knows that the packages and libraries help to execute the project most easily. because inside the libraries so many functions are executed at a time. The author displays all the libraries in Listing 2.1 and based on the Listing the author used Pandas to load the data.

```
import pandas as pd          # To deals with visualization and data loading
import numpy as np          # Handle numerical calculation and array values
import matplotlib.pyplot as plt # Deals with visualization
import plotly.express as px  # deal with advance interactive visualizations
import seaborn as sns       # deals with image related data
import cv2 as cv            # deals with train and train or model building in DL
import tensorflow as tf     # deal with models
from keras.preprocessing import image # to deals with image
from keras.models import Sequential # t run the model in a sequential way it is used
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout, BatchNormalization # importing different features to deal with
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os                   # change the directory and deal with operating systems
import warnings             # to avoid warning due to systems version issues
warnings.filterwarnings('ignore')
```

Listing 2.1 Load all the important Libraries to execute the program.

Similarly, the author also uses NumPy for handling all mathematical problems. For visualization purposes, the author used Matplotlib, Plotly, and Seaborn. For dealing with all the image data the author used a CV. With the help of Keras and TensorFlow, the author builds the model, all the libraries already mentioned in this Listing.

With the help of OS, the author loads different types of the class of image data. During the loading, the author used listdir from OS.

```
expression_class_name=os.listdir(r"C:\Users\manag\Downloads\dataset (1)\dataset\train") # Loading the classname of different i
expression_class_name
['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
```

Figure 2.1 Different types of expression name

This dataset contains seven different classes according to Figure 2.1 and all the classes are angry, sad, happy, neutral, surprised, fearful, and disgusted.

```
expression_class_name[0]
```

```
'angry'
```

Figure 2.2 shows the expression of class 0.

Now the author loading all seven classes one by one in this project. Also, the author shows the image based on content. In this case, the author gives the path of the dataset and with the help of a CV library, the author read the image data using the Imread function. After that, the author also shows all the images based on the classes

```
# Showing some of the images of angry class
plt.figure(figsize=(15,11))
path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[0]}" # Giving the dataset path of angry class
print(f"*****{expression_class_name[0]}*****")
for i in range(1,7):
    plt.subplot(3,3,i)
    plt.tight_layout()
    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xticks([])
    plt.yticks([])
```

Listing 2.2 Load the angry dataset.

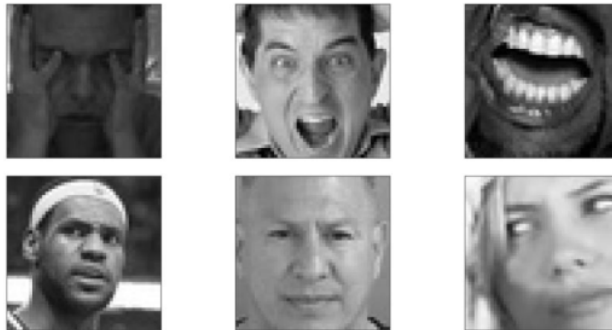


Figure 2.3 shows all the angry image which is present in the dataset.

```
# some of the images of disgust class
plt.figure(figsize=(15,11))
path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[1]}" # dataset path of disgust class
print(f"*****{expression_class_name[1]}*****")
for i in range(1,7):
    plt.subplot(3,3,i)
    plt.tight_layout()
    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xticks([])
    plt.yticks([])
```

Listing 2.3 Load disgusting dataset

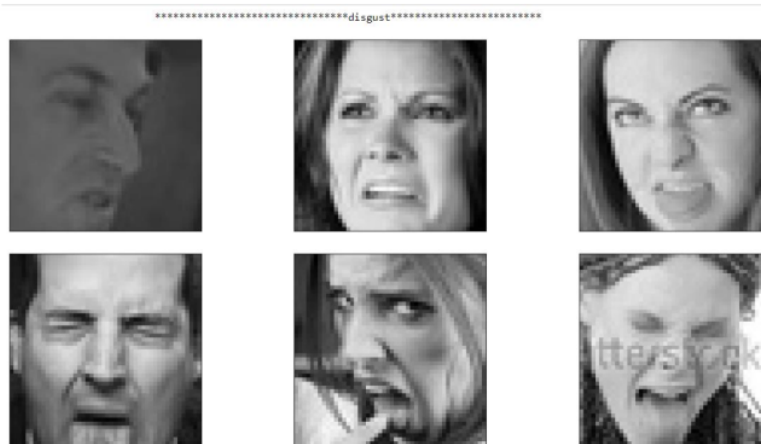


Figure 2.4 show the image data of disgusting which is available in the dataset

```
# Showing some images of fear class
plt.figure(figsize=(15,11))
path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[2]}" # data path of fear class
print(f"*****{expression_class_name[2]}*****")
for i in range(1,7):
    plt.subplot(3,3,i)
    plt.tight_layout()
    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xticks([])
    plt.yticks([])
```

Listing 2.4 Load fear image

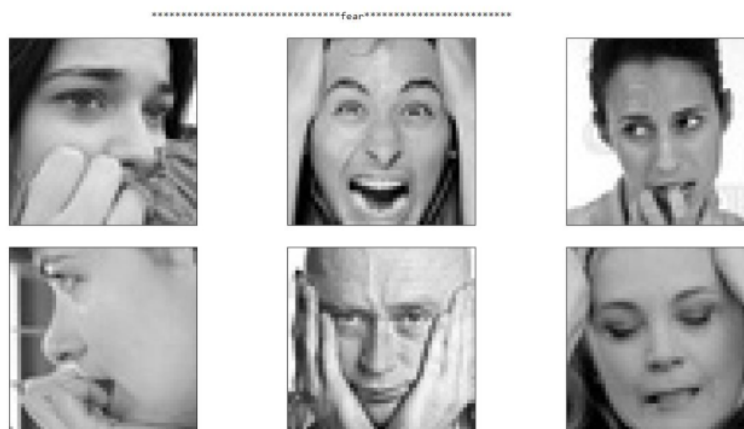


Figure 2.5 display the entire fear image which is present in this data.

[illegible]

Listing 2.5 loads the happy image

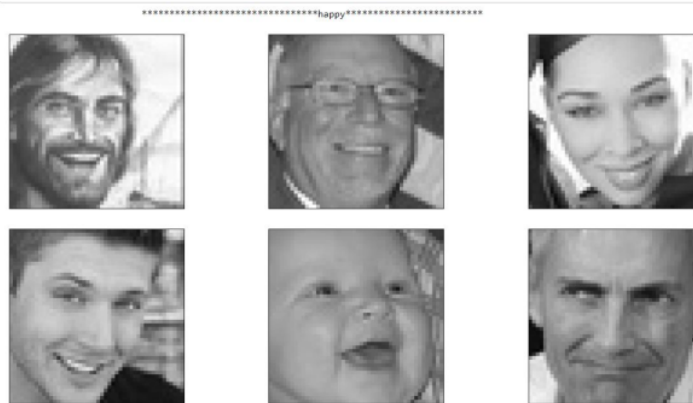


Figure 2.6 shows all the happy images

```
b7c<4>hexkey[0]
b7c<4>hexkey[1]
b7c<4>hexkey[2]
b7c<4>hexkey[3]
b7c<4>hexkey[4]
b7c<4>hexkey[5]
b7c<4>hexkey[6]
b7c<4>hexkey[7]
b7c<4>hexkey[8]
b7c<4>hexkey[9]
b7c<4>hexkey[10]
b7c<4>hexkey[11]
b7c<4>hexkey[12]
b7c<4>hexkey[13]
b7c<4>hexkey[14]
b7c<4>hexkey[15]
b7c<4>hexkey[16]
b7c<4>hexkey[17]
b7c<4>hexkey[18]
b7c<4>hexkey[19]
b7c<4>hexkey[20]
b7c<4>hexkey[21]
b7c<4>hexkey[22]
b7c<4>hexkey[23]
b7c<4>hexkey[24]
b7c<4>hexkey[25]
b7c<4>hexkey[26]
b7c<4>hexkey[27]
b7c<4>hexkey[28]
b7c<4>hexkey[29]
b7c<4>hexkey[30]
b7c<4>hexkey[31]
b7c<4>hexkey[32]
b7c<4>hexkey[33]
b7c<4>hexkey[34]
b7c<4>hexkey[35]
b7c<4>hexkey[36]
b7c<4>hexkey[37]
b7c<4>hexkey[38]
b7c<4>hexkey[39]
b7c<4>hexkey[40]
b7c<4>hexkey[41]
b7c<4>hexkey[42]
b7c<4>hexkey[43]
b7c<4>hexkey[44]
b7c<4>hexkey[45]
b7c<4>hexkey[46]
b7c<4>hexkey[47]
b7c<4>hexkey[48]
b7c<4>hexkey[49]
b7c<4>hexkey[50]
b7c<4>hexkey[51]
b7c<4>hexkey[52]
b7c<4>hexkey[53]
b7c<4>hexkey[54]
b7c<4>hexkey[55]
b7c<4>hexkey[56]
b7c<4>hexkey[57]
b7c<4>hexkey[58]
b7c<4>hexkey[59]
b7c<4>hexkey[60]
b7c<4>hexkey[61]
b7c<4>hexkey[62]
b7c<4>hexkey[63]
b7c<4>hexkey[64]
b7c<4>hexkey[65]
b7c<4>hexkey[66]
b7c<4>hexkey[67]
b7c<4>hexkey[68]
b7c<4>hexkey[69]
b7c<4>hexkey[70]
b7c<4>hexkey[71]
b7c<4>hexkey[72]
b7c<4>hexkey[73]
b7c<4>hexkey[74]
b7c<4>hexkey[75]
b7c<4>hexkey[76]
b7c<4>hexkey[77]
b7c<4>hexkey[78]
b7c<4>hexkey[79]
b7c<4>hexkey[80]
b7c<4>hexkey[81]
b7c<4>hexkey[82]
b7c<4>hexkey[83]
b7c<4>hexkey[84]
b7c<4>hexkey[85]
b7c<4>hexkey[86]
b7c<4>hexkey[87]
b7c<4>hexkey[88]
b7c<4>hexkey[89]
b7c<4>hexkey[90]
b7c<4>hexkey[91]
b7c<4>hexkey[92]
b7c<4>hexkey[93]
b7c<4>hexkey[94]
b7c<4>hexkey[95]
b7c<4>hexkey[96]
b7c<4>hexkey[97]
b7c<4>hexkey[98]
b7c<4>hexkey[99]
```

Listing 2.6 load neutral images

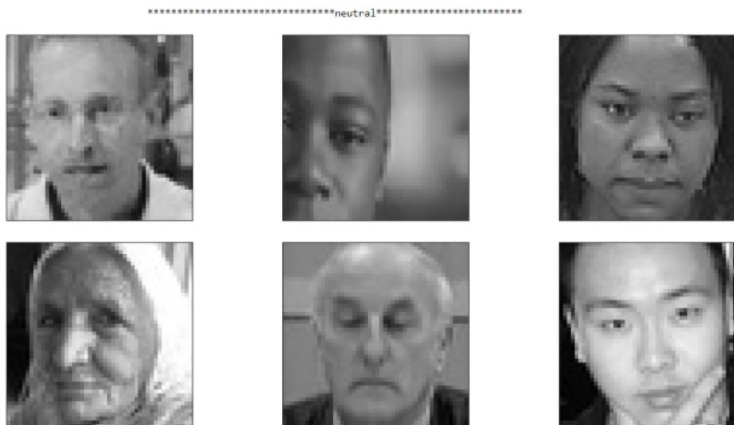


Figure 2.7 demonstrates all the neutral images

```
# Showing some images of sad class
plt.figure(figsize=(15,11))
path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[5]}" # data path of sad class
print(f"*****{expression_class_name[5]}*****")
for i in range(1,7):
    plt.subplot(3,3,i)
    plt.tight_layout()
    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xticks([])
    plt.yticks([])
```

Listing 2.7 loads sad images

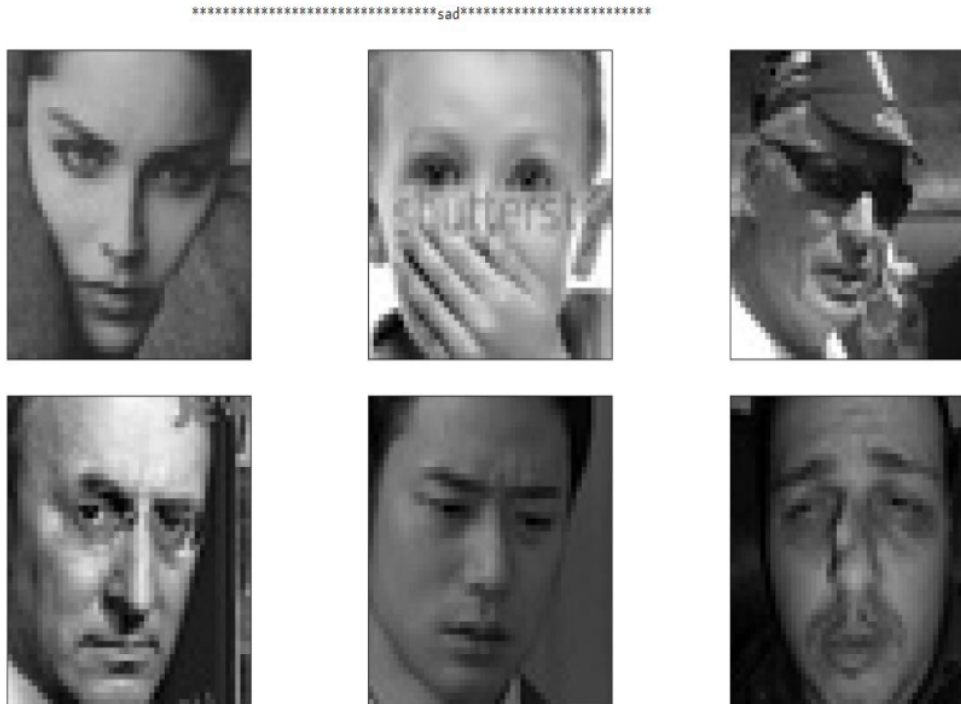


Figure 2.8 display all the sad image which is present in this dataset.

```
# Showing some images of surprise class
plt.figure(figsize=(15,11))
path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[6]}" # data path of surprise class
print(f"*****{expression_class_name[6]}*****")
for i in range(1,7):
    plt.subplot(3,3,i)
    plt.tight_layout()
    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xticks([])
    plt.yticks([])
```

Listing 2.8 surprise image loading with the help of cv

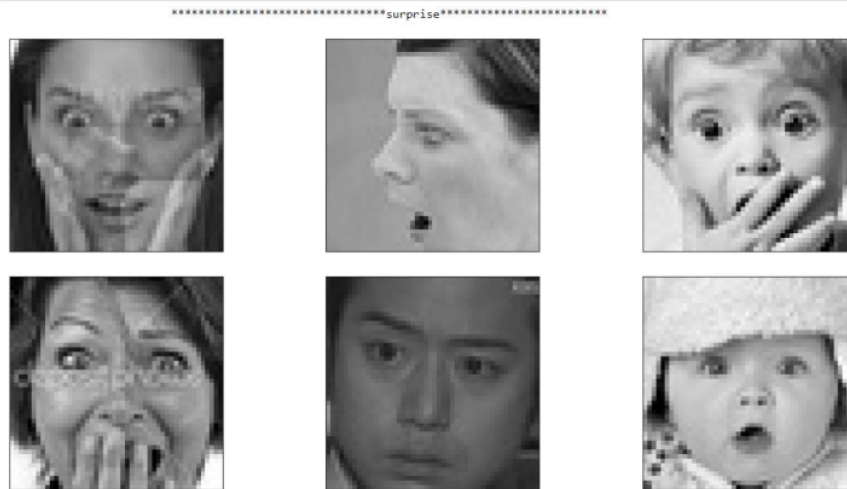


Figure 2.9 shows all the surprise pictures

After that, the author creates a plot for getting more information about the dataset. Because data visualization helps extract many hidden information which helps for increasing the accuracy as well as properly create the model.

```
colors = ['#1b9e77', '#a9f971', '#fdaa48', '#6890f0', '#a890f0', 'red', 'yellow']
count_of_different_class=[]
for i in Class_name:
    count_of_different_class.append(len(os.listdir("C:/Users/manag/Downloads/dataset (1)/dataset/train/"+i+"/")))
plt.bar(Class_name,count_of_different_class,color=colors)
plt.xticks(rotation=45);
```

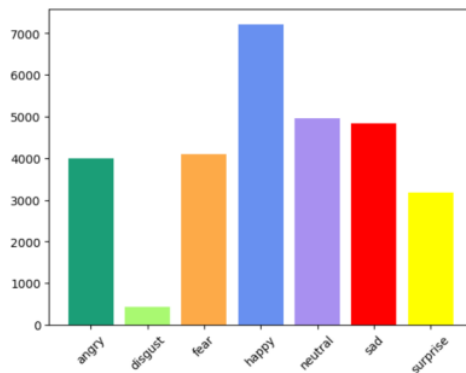


Figure 2.10 display all the class for comparing one class to another class.

With the help of Matplotlib, the author creates a one-bar plot that helps for comparing one class to another class. According to Figure 2.10, the author found that the class “happy” has more data and the class “disgust” has less amount of data. Also, the author found that class

“angry” and class “fear” has an equal amount of data. It means a greater number of people are happy based on the dataset.

After that, the author divides the whole data into training data and testing data because, in every data science project, it must be important the train model properly. If the model was properly trained it easily gives more predictive outcomes.

Through Imagedata Generator generating more data

```
train_datagen = ImageDataGenerator(
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    horizontal_flip = True,
    rescale = 1./255,
    zoom_range = 0.2,
    validation_split = 0.2
)

test_datagen = ImageDataGenerator(rescale = 1./255,
)

train_generator = train_datagen.flow_from_directory(directory = "C:/Users/manag/Downloads/dataset (1)/dataset/train/",
    target_size = (48,48),
    batch_size = 32,
    color_mode = "grayscale",
    class_mode = "categorical"
)

test_generator = test_datagen.flow_from_directory(directory = "C:/Users/manag/Downloads/dataset (1)/dataset/test",
    target_size = (48,48),
    batch_size = 32,
    color_mode = "grayscale",
    class_mode = "categorical",
)
```

Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.

Figure 2.11 divide all the dataset into training data and testing data.

Here the author divides the entire data into 80 % and 20%. And 20% of the data will store to test the model and the remaining data may used to train the model. The author found that 28709 data are stored for training purposes and the remaining 7178 data will save for testing purposes but in both cases, the class will be equal.

Then the author designs one model architecture to execute the program. During this time the author used CNN techniques from DL. Because it is a popular technique and is capable to handle all the image data.

Designing the model architecture of the proposed work

```
model= Sequential() # Loading the sequential process of model
model.add(Conv2D(32,(3,3),padding='same',activation='relu',input_shape=(48,48,1))) # Adding activation function and input shape
model.add(Conv2D(64,(3,3),padding='same',activation='relu')) # Adding different hidden Layer
model.add(BatchNormalization()) # To learn the pattern more independently
model.add(MaxPool2D(pool_size=(2, 2))) # To reduce the dimentionalty of images
model.add(Dropout(0.25)) # Apply dropout to avoid overfitting

model.add(Conv2D(128,(3,3),padding='same',activation='relu')) # Adding more hidden Layer
model.add(Conv2D(256,(3,3),padding='same',activation='relu'))
model.add(BatchNormalization()) # To learn the pattern more independently
model.add(MaxPool2D(pool_size=(2, 2))) # To reduce the dimentionalty of images
model.add(Dropout(0.25)) # Apply dropout to avoid overfitting

model.add(Conv2D(256,(3,3),padding='same',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(100,activation = 'relu')) # Adding relu because it is robust to the vanishing gradient problem
model.add(Dense(50,activation = 'relu'))
model.add(Dense(225,activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Dense(7, activation='softmax')) # Adding softmax in output layer because probability distribution of classes
```

Listing 2.9 Creating the model using CNN

Before creating the model, the author called the Sequential function, and based on this function the author used “relu” as an activation function in the input layer and for the output layer the author used “softmax”.

Then the author also shows the summary of the creating model. Here shows the name of the Layers, the shape of the output, and the total number of Param.

```

: model.summary() # showing the summary of the designed model
Model: "sequential"

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	320
conv2d_1 (Conv2D)	(None, 48, 48, 64)	18496
batch_normalization (Batch Normalization)	(None, 48, 48, 64)	256
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0
dropout (Dropout)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	73856
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295168
batch_normalization_1 (Batch Normalization)	(None, 24, 24, 256)	1024
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 256)	0
dropout_1 (Dropout)	(None, 12, 12, 256)	0
conv2d_4 (Conv2D)	(None, 12, 12, 256)	590880
batch_normalization_2 (Batch Normalization)	(None, 12, 12, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
dropout_2 (Dropout)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 100)	921700
dense_1 (Dense)	(None, 50)	5050
dense_2 (Dense)	(None, 225)	11475
batch_normalization_3 (Batch Normalization)	(None, 225)	900
dropout_3 (Dropout)	(None, 225)	0
dense_3 (Dense)	(None, 7)	1582

```

Total params: 1,920,931
Trainable params: 1,919,329
Non-trainable params: 1,602

```

Figure 2.12 shows the summary of the model.

The total number of Params are 1920931 and based on the total params, the total trainable params are 1919329 and the non-trainable params are 1602 according to Figure 2.12.

After that, the author compiles the entire model using categorical cross entropy and the techniques shown in Listing 2.10.

```

model.compile(
    optimizer=tf.keras.optimizers.Adam(lr=0.0005),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

```

WARNING:absl:'lr' is deprecated in Keras optimizer, please use 'learning_rate' or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.

Listing 2.10 compile the model with the help of Adam optimizers

Here the author fits the model with train data for teaching the model and during this process, the author used 55 epochs and the batch size is 32.

```

history = model.fit(
    train_generator ,
    epochs=55,
    batch_size=32)

```

Epoch 1/55
 898/898 [=====] - 575s 637ms/step - loss: 1.9220 - accuracy: 0.2344
 Epoch 2/55
 898/898 [=====] - 586s 653ms/step - loss: 1.6547 - accuracy: 0.3450
 Epoch 3/55
 898/898 [=====] - 552s 615ms/step - loss: 1.4614 - accuracy: 0.4325
 Epoch 4/55
 898/898 [=====] - 541s 602ms/step - loss: 1.3413 - accuracy: 0.4879
 Epoch 5/55
 898/898 [=====] - 486s 542ms/step - loss: 1.2848 - accuracy: 0.5133
 Epoch 6/55
 898/898 [=====] - 535s 596ms/step - loss: 1.2358 - accuracy: 0.5340
 Epoch 7/55
 898/898 [=====] - 550s 613ms/step - loss: 1.1985 - accuracy: 0.5484
 Epoch 8/55
 898/898 [=====] - 498s 555ms/step - loss: 1.1778 - accuracy: 0.5566
 Epoch 9/55
 898/898 [=====] - 485s 541ms/step - loss: 1.1504 - accuracy: 0.5673
 Epoch 10/55
 898/898 [=====] - 476s 530ms/step - loss: 1.1324 - accuracy: 0.5742
 Epoch 11/55
 898/898 [=====] - 508s 566ms/step - loss: 1.1154 - accuracy: 0.5817
 Epoch 12/55
 898/898 [=====] - 560s 624ms/step - loss: 1.0895 - accuracy: 0.5927
 Epoch 13/55
 898/898 [=====] - 556s 619ms/step - loss: 1.0709 - accuracy: 0.6010
 Epoch 14/55
 898/898 [=====] - 722s 804ms/step - loss: 1.0640 - accuracy: 0.6031
 Epoch 15/55
 898/898 [=====] - 774s 862ms/step - loss: 1.0445 - accuracy: 0.6086
 Epoch 16/55
 898/898 [=====] - 775s 863ms/step - loss: 1.0312 - accuracy: 0.6163
 Epoch 17/55
 898/898 [=====] - 690s 769ms/step - loss: 1.0222 - accuracy: 0.6197
 Epoch 18/55

Figure 2.13 shows the first epochs

According to this Figure 2.13, the author got 61.97 accuracies in this stage and the epoch number is 18. But if anyone observed this figure the first epochs give very less accuracy 23.44 % and after 18 epochs it will increase and goes to 61.97. it means that this model learns something new in every epoch.

```

Epoch 34/55
898/898 [=====] - 516s 574ms/step - loss: 0.8803 - accuracy: 0.6746
Epoch 35/55
898/898 [=====] - 516s 575ms/step - loss: 0.8781 - accuracy: 0.6737
Epoch 36/55
898/898 [=====] - 571s 636ms/step - loss: 0.8729 - accuracy: 0.6759
Epoch 37/55
898/898 [=====] - 593s 660ms/step - loss: 0.8570 - accuracy: 0.6821
Epoch 38/55
898/898 [=====] - 528s 588ms/step - loss: 0.8571 - accuracy: 0.6837
Epoch 39/55
898/898 [=====] - 576s 641ms/step - loss: 0.8515 - accuracy: 0.6862
Epoch 40/55
898/898 [=====] - 561s 625ms/step - loss: 0.8469 - accuracy: 0.6855
Epoch 41/55
898/898 [=====] - 560s 623ms/step - loss: 0.8366 - accuracy: 0.6930
Epoch 42/55
898/898 [=====] - 556s 619ms/step - loss: 0.8460 - accuracy: 0.6868
Epoch 43/55
898/898 [=====] - 543s 605ms/step - loss: 0.8268 - accuracy: 0.6943
Epoch 44/55
898/898 [=====] - 567s 631ms/step - loss: 0.8205 - accuracy: 0.6970
Epoch 45/55
898/898 [=====] - 568s 632ms/step - loss: 0.8219 - accuracy: 0.6976
Epoch 46/55
898/898 [=====] - 555s 619ms/step - loss: 0.8120 - accuracy: 0.6993
Epoch 47/55
898/898 [=====] - 519s 578ms/step - loss: 0.8087 - accuracy: 0.7015
Epoch 48/55
898/898 [=====] - 485s 540ms/step - loss: 0.8027 - accuracy: 0.7058
Epoch 49/55
898/898 [=====] - 521s 580ms/step - loss: 0.8011 - accuracy: 0.7056
Epoch 50/55
898/898 [=====] - 633s 705ms/step - loss: 0.7855 - accuracy: 0.7102
Epoch 51/55
898/898 [=====] - 775s 863ms/step - loss: 0.7968 - accuracy: 0.7068
Epoch 52/55
898/898 [=====] - 770s 857ms/step - loss: 0.7802 - accuracy: 0.7154
Epoch 53/55
898/898 [=====] - 673s 749ms/step - loss: 0.7778 - accuracy: 0.7146
Epoch 54/55
898/898 [=====] - 507s 564ms/step - loss: 0.7717 - accuracy: 0.7160
Epoch 55/55
898/898 [=====] - 503s 560ms/step - loss: 0.7692 - accuracy: 0.7165

```

Figure 2.14 shows the last epochs.

The author got 71.65 accuracy when the epoch number is 55. It means that if the number of epochs is increased then automatically the accuracy also increases.

```

model.evaluate(test_generator) # Finally check the model performance through accuracy evaluation matrix
225/225 [=====] - 38s 151ms/step - loss: 0.9496 - accuracy: 0.6673
[0.9496102333068848, 0.6673167943954468]

```

Figure 2.15 evaluate the model

Here evaluates the model with the help of a test generator and it gives 66.73% accuracy.

Accuracy and Loss values VS Number of Epochs in the model

```
train_acc = history.history['accuracy']
train_loss = history.history['loss']

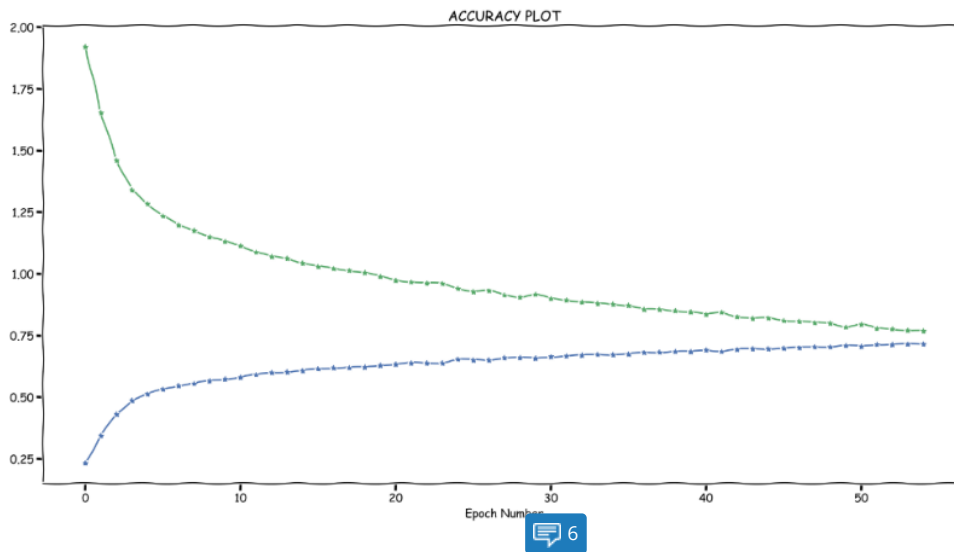
plt.xkcd()
plt.style.use('seaborn-deep')

plt.rcParams["figure.figsize"] = (20, 10)

plt.plot(history.history['accuracy'],marker="*",label="Acc Value")
plt.plot(history.history['loss'],marker="*",label="loss Value")
plt.grid(True)

plt.title("ACCURACY PLOT")
plt.xlabel("Epoch Number")
plt.ylabel("")
```

Listing 2.11 create the plot for showing the train accuracy and test accuracy.



Here the author shows the accuracy plot of this model, basically, here the author tries to compare the test accuracy and train accuracy based on epoch number

Figure 2.16 display the training and testing accuracy graphically.

```
] : model.save("My_model.h5") # Saving the models
    model.save_weights("Model_weights.h5") # saving the model weights
```

Listing 2.12 save the model.

After completing all the steps, the author finally saves this model because, in the DL model, it takes so much time when the execute the model. To avoid this problem, the author saves this model as well as saves the weight of the model in a better way.

3 Conclusion, Recommendation, and Future Scope

3.1 Conclusion

Here, after implementing the CNN model architecture the author applies model evaluation techniques to find the model performance, and for that the author used Accuracy evaluation techniques as it is a classification problem. The author got approx. 66.73 percent of accuracy with this model and 0.94 as loss value. Here the author creates only one model because computing the model takes a huge amount of time as it processes the data and then learns the pattern and then gives the output as accuracy. Because the time consumption is more than that's why only one model was created but this model the author cannot consider this the best model because the accuracy is only 66.73 but it is not as much worst performance also. Here the author used only 55 epochs but generally, it is considered 100 epochs to be the best to train the model by using 55 epochs also the author faced a lot of time taken issues. that's why the author finalizes this model as the best suitable model for this problem statement for this work-study.

3.2 Recommendation

As already mentioned in the conclusion in section 3.1 that it takes more time to compute or learn all the patterns, so it is recommended to use a server to run the program instead of using Jupyter and colab notebook. Apart from that as the accuracy is low due to fewer epochs so it is suggested to use at least 100 epochs to get a more accurate result.

3.3 Future Scope

As the model is capable to predict human emotions from facial expressions so there is a huge chance in the upcoming future. It can be useful in an online class for the students because while the teacher taking classes some people are interested and some are not so, in that case by catching the emotions if a person is not interested or busy with other things gives the alert to both students and teachers so that teachers can pay more attention for those students. This is one application or scope for that work study. Apart from that the same thing can be applied in a business meeting etc to get the emotions behind the person's behaviors.

4 References

- [1] Sanika. D, C. Wessels, V. Kanal, C. Sevastopoulos, A. Jaiswal, H. Nambiappan, and F. Makedon. "A review of extended reality (xr) technologies for manufacturing training." *Technologies* 8, no. 4 (2020): 77.
- [2] Anirudha. G, A. Sufian, F. Sultana, A. Chakrabarti, and D. De. "Fundamental concepts of convolutional neural network." *Recent trends and advances in artificial intelligence and Internet of Things* (2020): 519-567.
- [3] Dickshon. H.N, M. A. Hannan, M. S. H. Lipu, K. S. Sahari, P. J. Ker, and K. M. Muttaqi. "State-of-charge estimation of li-ion battery in electric vehicles: A deep neural network approach." *IEEE Transactions on Industry Applications* 56, no. 5 (2020): 5565-5574.
- [4] Nei. K, Z. M. Fadlullah, F. Tang, B. Mao, S. Tani, A. Okamura, and J. Liu. "Optimizing space-air-ground integrated networks by artificial intelligence." *IEEE Wireless Communications* 26, no. 4 (2019): 140-147.
- [5] Cadena E. M, L. F. P. Martínez, and J. E. O. Triviño. "Detection of malicious primary user emulation based on a support vector machine for a mobile cognitive radio network using software-defined radio." *Electronics* 9, no. 8 (2020): 1282.
- [6] Shouxiang. W, and H. Chen. "A novel deep learning method for the classification of power quality disturbances using deep convolutional neural network." *Applied energy* 235 (2019): 1126-1140.
- [7] Ebanesar, T., A. D. Bibin, and J. Jalaja. "HUMAN EAR RECOGNITION USING CONVOLUTIONAL NEURAL NETWORK." *Journal of Positive School Psychology* (2022): 8182-8190.
- [8] Guoqing. X, J. Li, Y. Chen, and K. Li. "MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks." *Journal of Parallel and Distributed Computing* 141 (2020): 49-58.
- [9] Laura. R, J. Andreas, M. Baroni, D. Bouchacourt, and B. M. Lake. "A benchmark for systematic generalization in grounded language understanding." *Advances in neural information processing systems* 33 (2020): 19861-19872.

5 Appendices

```
### Importing Libraries

import pandas as pd          # To deals with visualization and data loading

import numpy as np           # Handle numerical calculation and array values

import matplotlib.pyplot as plt # Deals with visualization

import plotly.express as px   # deal with advance interactive visualizations

import seaborn as sns

import cv2 as cv             # deals with image related data

import tensorflow as tf       # deals with train and train or model building in DL

import keras                 # deal with models

from keras.preprocessing import image # to deals with image

from keras.models import Sequential # t run the model in a sequential way it is used

from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout, BatchNormalization #
importing different features to deal with model buildiung

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os # change the directory and deal with operating systems

import warnings # to avoid warning due to systems version issues

warnings.filterwarnings('ignore')

!pip install opencv-python

### Explore data

expression_class_name=os.listdir(r"C:\Users\manag\Downloads\dataset (1)\dataset\train") #
Loading the classname of different image class

expression_class_name

expression_class_name[0]

#### Observation:
```

- It can be seen that there are total number of seven expression class available.

Showing different image class of various facial Expression

Showing some of the images of angry class

```
plt.figure(figsize=(15,11))
```

```
path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[0]}" #
```

Giving the dataset path of angry class

```
print(f"
```

```
*****{expression_class_name[0]}*****
*****")
```

```
for i in range(1,7):
```

```
    plt.subplot(3,3,i)
```

```
    plt.tight_layout()
```

```
    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))
```

```
    plt.imshow(rand_img)
```

```
    plt.xticks([])
```

```
    plt.yticks([])
```

some of the images of disgust class

```
plt.figure(figsize=(15,11))
```

```
path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[1]}" #
```

dataset path of disgust class

```
print(f"
```

```
*****{expression_class_name[1]}*****
*****")
```

```
for i in range(1,7):
```

```
    plt.subplot(3,3,i)
```

```

plt.tight_layout()

rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))

plt.imshow(rand_img)

plt.xticks([])

plt.yticks([])

# Showing some images of fear class

plt.figure(figsize=(15,11))

path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[2]}" #
data path of fear class

print(f"
*****{expression_class_name[2]}*****
****")

for i in range(1,7):

    plt.subplot(3,3,i)

    plt.tight_layout()

    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))

    plt.imshow(rand_img)

    plt.xticks([])

    plt.yticks([])

# Showing some images of happy class

plt.figure(figsize=(15,11))

path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[3]}" #
data path of happy class

print(f"
*****{expression_class_name[3]}*****
****")

```

```

for i in range(1,7):

    plt.subplot(3,3,i)

    plt.tight_layout()

    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))

    plt.imshow(rand_img)

    plt.xticks([])

    plt.yticks([])

# Showing some images of neutral class

plt.figure(figsize=(15,11))

path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[4]}" #
data path of neutral class

print(f"
*****{expression_class_name[4]}*****
****")

for i in range(1,7):

    plt.subplot(3,3,i)

    plt.tight_layout()

    rand_img = cv.imread(path + '/' + np.random.choice(sorted(os.listdir(path))))

    plt.imshow(rand_img)

    plt.xticks([])

    plt.yticks([])

# Showing some images of sad class

plt.figure(figsize=(15,11))

path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[5]}" #
data path of sad class

```

```

print(f"
*****{expression_class_name[5]}*****
****")

for i in range(1,7):

    plt.subplot(3,3,i)

    plt.tight_layout()

    rand_img = cv.imread(path +'/' + np.random.choice(sorted(os.listdir(path))))

    plt.imshow(rand_img)

    plt.xticks([])

    plt.yticks([])


# Showing some images of surprise class

plt.figure(figsize=(15,11))

path = f"C:/Users/manag/Downloads/dataset (1)/dataset/train/{expression_class_name[6]}" #
data path of surprise class

print(f"
*****{expression_class_name[6]}*****
****")

for i in range(1,7):

    plt.subplot(3,3,i)

    plt.tight_layout()

    rand_img = cv.imread(path +'/' + np.random.choice(sorted(os.listdir(path))))

    plt.imshow(rand_img)

    plt.xticks([])

    plt.yticks([])


# Showing the distribution of training data

```

```

colors = ['#1b9e77', '#a9f971', '#fdaa48', '#6890F0', '#A890F0', 'red', 'yellow']

count_of_different_class=[]

for i in Class_name:

    count_of_different_class.append(len(os.listdir("C:/Users/manag/Downloads/dataset
(1)/dataset/train/"+i+"/")))

plt.bar(Class_name,count_of_different_class,color=colors)

plt.xticks(rotation=45);

count_of_different_class

#### Observation:

- It is observed that there are happy classes are more in number.

- After that neutral and sad and disgust are very less compare to other emotions expression.

# Through Imagedata Generator generating more data

train_datagen = ImageDataGenerator(

    width_shift_range = 0.1,

    height_shift_range = 0.1,

    horizontal_flip = True,

    rescale = 1./255,

    zoom_range = 0.2,

    validation_split = 0.2)

test_datagen = ImageDataGenerator(rescale = 1./255,

    )

train_generator = train_datagen.flow_from_directory(directory =

"C:/Users/manag/Downloads/dataset (1)/dataset/train/",

    target_size = (48,48),

    batch_size = 32,

```

```

        color_mode = "grayscale",

        class_mode = "categorical"

    )

test_generator = test_datagen.flow_from_directory(directory =
"C:/Users/manag/Downloads/dataset (1)/dataset/test",

        target_size = (48,48),

        batch_size = 32,

        color_mode = "grayscale",

        class_mode = "categorical",

    )

# Designing the model architecture of the proposed work

model= Sequential() # loading the sequential process of model

model.add(Conv2D(32,(3,3),padding='same',activation='relu',input_shape=(48,48,1))) #
Adding activation function and input shape

model.add(Conv2D(64,(3,3),padding='same',activation='relu')) # Adding different hidden
layer

model.add(BatchNormalization()) # To learn the pattern more independently

model.add(MaxPool2D(pool_size=(2, 2))) # To reduce the dimentionality of images

model.add(Dropout(0.25)) # Apply dropout to avoid overfitting

model.add(Conv2D(128,(3,3),padding='same',activation='relu')) # Adding more hidden layer

model.add(Conv2D(256,(3,3),padding='same',activation='relu'))

model.add(BatchNormalization()) # To learn the pattern more independently

model.add(MaxPool2D(pool_size=(2, 2))) # To reduce the dimentionality of images

model.add(Dropout(0.25)) # Apply dropout to avoid overfitting

model.add(Conv2D(256,(3,3),padding='same',activation='relu'))

```

```

model.add(BatchNormalization())

model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(100,activation = 'relu')) # Adding relu because it is robust to the vanishing
gradient problem

model.add(Dense(50,activation = 'relu'))

model.add(Dense(225,activation = 'relu'))

model.add(BatchNormalization())

model.add(Dropout(0.25))

model.add(Dense(7, activation='softmax')) # Adding softmax in output layer because
probability distribution of classes

model.summary() # showing the summary of the designed model

## Model Compilation with adam, loss function and accuracy evaluation matrix

model.compile(

    optimizer = tf.keras.optimizers.Adam(lr=0.0005),

    loss='categorical_crossentropy',

    metrics=['accuracy']

)

## Model fitting or training with train data

history = model.fit(

    train_generator ,

    epochs=55,

    batch_size=32)

```

```

model.evaluate(test_generator) # Finally check the model performance through accuracy
evaluation matrix

# Accuracy and Loss values VS Number of Epochs in the model

train_acc = history.history['accuracy']

train_loss = history.history['loss']

plt.xticks()

plt.style.use('seaborn-deep')

plt.rcParams["figure.figsize"] = (20, 10)

plt.plot(history.history['accuracy'],marker="*",label="Acc Value")

plt.plot(history.history['loss'],marker="*",label="loss Value")

plt.grid(True)

plt.title("ACCURACY PLOT")

plt.xlabel("Epoch Number")

plt.ylabel("")

plt.show()

## saving the trained model

model.save("My_model.h5") # Saving the models

model.save_weights("Model_weights.h5") # saving the model weights

```

6 Video link

<https://youtu.be/VUSWprd2PT0>

Application of Machine Learning Assignment 2

GRADEMARK REPORT

FINAL GRADE

59/100

GENERAL COMMENTS

Instructor

Nice to see table of contents, figures, etc.

No abstract.

Writing style could be a little more formal - occasionally it needs to be a little more terse. In places you include comments related to a specific context (eg radiology). We're not working with radiology data.

Avoid writing lists - eg advantages vs disadvantages. Whilst it is important to include a discussion of these, apply a more sophisticated style when analysing. Eg place the adv /dis-adv. in a problem context.

When reporting on your design please explain WHY you selected certain values, etc - eg refer to related work that informed your choices.

Your results show accuracy of your models but you need to include confusion matrices and analyse the classes -- eg explain where/why misclassifications might appear.

Use additional metrics - recall, F1,etc here to help better understand the results and so draw appropriate conclusions.

PAGE 1

PAGE 2

PAGE 3



Comment 1

Figures - note plural.



Comment 2

Listings

PAGE 4

PAGE 5

PAGE 6



Comment 3

too specific a context. We're not working with radiology data.

PAGE 7



Comment 4

Good. You have a reason to use these and you have identified it.

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15



Comment 5

Validation data?

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20



Comment 6

label your axes and provide a key!

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

GRADING FORM: SMU GENERIC_GJ

GOWRI PAMARTHI

0

RESEARCH-INFORMED LITERATURE

KNOWLEDGE AND UNDERSTANDING

ANALYSIS

PRACTICAL APPLICATION AND DEP.

SKILLS FOR PROFESSIONAL PRAC.