REAL-TIME EMOTION DETECTION MODEL

Submitted by: Misha Dey

DESCRIPTION:

Our Approach is to detect the facial emotions using the Facial Action Units.

FACS(Facial action coding system):

Facial Muscular Movements causing momentary changes in the facial expressions are termed as **AU(Action Units)**

Facial Expressions are identified by using a single Action unit or a series of action units which are associated with facial movements.

DATASET DESCRIPTION:

Dataset:

https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

We are using 48x48 pixel grayscale images of faces

Data Classified into -> seven categories:

1. **0**=Angry
2. **1**=Disgust
3. **2**=Fear
4. **3=**Happy
5. **4**=Sad
6. **5**=Surprise
7. **6**=Neutral

Columns: "emotion" and "pixels"

No. of Classes: 7 (7 Emotions)

TOOLS:

1. Keras

2. TensorFlow

3. OpenCV

4. Pandas

5. Numpy

6. Flask

7. Pillow

8. Gunicorn

Training Data: 28,709 samples.

Testing Data: 7178 samples

PROJECT DIRCTORY: Real Time Emotion Detection from Facial Expression using CNN

Link: https://drive.google.com/drive/folders/161Yi5qqLs9jBsGJGNPY54eKlwmSE8KiJ

Project Directory Structure:

0. **Data:**

--> test.csv
--> train.csv

1. **Training Model Version 1:**

--> Brightness_And_Sharpness_Augmented_data2.ipynb

--> CNN_Model.py

--> CNN_Model1.h5

--> CNN_Model_Epochs.csv

--> CNN_Model_Final_v1.h5

--> CNN_Model_Plot.png

--> Data_Preprocessing.py

--> Real_Time_Emotion_Detection_from_using_CNN_Version1.ipynb

--> Utils_funX.py

--> test.csv (Now in the Data Directory)

--> train.csv (Now in the Data Directory)

2. Realtime Webcam Prediction Version 1 (Deploying App on Localhost)

--> app.py

--> Real_Time_Webcam_Demonstration.ipynb

--> CNN_Model_Final_v1.h5

--> Real_Time_Webcam_Demonstration.py

--> Recorded_Time_Webcam_Demonstration.py

--> image_tools.py

--> Utils_funX.py

--> opencv-dnn

--> deploy.prototxt

--> weights.caffemodel

3. Realtime Webcam Prediction Version 2 (Deploying App on Heroku)

--> app.py

--> CNN_Model_Final_v1.h5

--> Real_Time_Webcam_Demonstration.py

--> requirements.txt

--> image_tools.py

--> Utils_funX.py

--> deploy.prototxt

--> weights.caffemodel

--> Aptfile

--> Procfile

--> runtime.txt

DATA PRE-PROCESSING AND AUGMENTATION:

I cleaned the data and removed the duplicates and augmented the data to create 'vertically augmented', 'sharpness augmented' and 'brightness augmented' data.

This task is done by two modules:

Data_preprocessing.py

```
import numpy as np
from sklearn.preprocessing import normalize
from plantcv import plantcv as pcv
import matplotlib.pyplot as plt
from Utils_funX import *

def Data_Preparation(train_data,test_data):
    x_train,y_train,x_test,y_test=[],[],[],[]
    # pixel values are in x_values and the categories/clases are the y_values
    for index,row in train_data.iterrows():
        row_val=np.array(row['pixels'].split(" "))
        x_train.append(np.array(row_val,'float64'))
    x_train = np.array(x_train,'float64')
    y_train = train_data["emotion"].values

    for index,row in test_data.iterrows():
        row_val=np.array(row['pixels'].split(" "))
        x_test.append(np.array(row_val,'float64'))
    x_test = np.array(x_test,'float64')
    y_test=np.array(y_test)

    #y_test=y_test.reshape((y_test.shape[0],1))
    #y_train=y_train.reshape((y_train.shape[0],1))
    return x_train,y_train,x_test,y_test

def Data_Augmentation(x_train,y_train):
    # mirroring the train data() along the vertical and horozontal Axis
    # Concept --> Only the x_train will be changed --> that is the pixel values will be mirrored
    # the y_train values will remain same --> since the labels coressponding to each emotion will remail same
    for flip_type in ['vertical']:
        augmented_x_train = []
        augmented_y_train = []

        for i,image in enumerate(x_train):
            augmented_image,augmented_labels = get_mirrored_data(image,y_train[i],flip_type)
            augmented_x_train.append(augmented_image)
            augmented_y_train.append(augmented_labels)

        print("\nFive Sample of image after {} flip :\n".format(flip_type))
        for i in range(5):
            plt.imshow(augmented_x_train[i])
        for i in range(5):
            plt.imshow(augmented_x_train[i])
            plt.title(Decode_Y_Val(augmented_y_train[i]))
            plt.show()

        x_train = np.concatenate((x_train,augmented_x_train))
        y_train = np.concatenate((y_train,augmented_y_train))

    return x_train,y_train

def get_mirrored_data(image_pixel,label,flip_type):
    aug_img = pcv.flip(image_pixel,flip_type)
    aug_label = label
    return aug_img,aug_label


def Data_Normalization(x_train,x_test):
    x_train = normalize(x_train)
    x_test = normalize(x_test)
    return x_train,x_test
```

Brightness_And_Sharpness_Augmented_data2.ipynb

```
12
13      # adding the augmented data with increased brightness to the new_data_sett
14      for i in range(data_set_len):
15          aug_img = enhance_image_brighness(x_train[i],'increase')
16          inc_brighteness_x.append(aug_img)
17          inc_brighteness_y.append(y_train[i])
18      new_x_data = np.array(inc_brighteness_x)
19      new_y_data = np.array(inc_brighteness_y)
20
21      # adding the augmented data with decreased brightness to the new_data_sett
22      for i in range(data_set_len):
23          aug_img = enhance_image_brighness(x_train[i],'decrease')
24          dec_brighteness_x.append(aug_img)
25          dec_brighteness_y.append(y_train[i])
26      new_x_data = np.concatenate((new_x_data,dec_brighteness_x))
27      new_y_data = np.concatenate((new_y_data,dec_brighteness_y))
28
29      # adding the augmented data with increased sharpness to the new_data_set
30      for i in range(data_set_len):
31          aug_img = enhance_image_sharpness(x_train[i],'increase')
32          inc_sharpness_x.append(aug_img)
33          inc_sharpness_y.append(y_train[i])
34      new_x_data = np.concatenate((new_x_data,inc_sharpness_x))
35      new_y_data = np.concatenate((new_y_data,inc_sharpness_y))
36
37      # adding the augmented data with increased brightness to the new_data_sett
38      for i in range(data_set_len):
39          aug_img = enhance_image_sharpness(x_train[i],'decrease')
40          dec_sharpness_x.append(aug_img)
41          dec_sharpness_y.append(y_train[i])
42      new_x_data = np.concatenate((new_x_data,dec_sharpness_x))
43      new_y_data = np.concatenate((new_y_data,dec_sharpness_y))
44
45      return new_x_data,new_y_data
```

```
1  from PIL import Image,ImageEnhance
2  from keras.preprocessing.image import array_to_img
3
4  def enhance_image_brighness(image_array,operation):
5      img = array_to_img(np.reshape(image_array,(48,48,1)))
6
7      enhancer = ImageEnhance.Brightness(img)
8      if(operation == 'increase'):
9          enhanced_img = enhancer.enhance(1.3)
10     else:
11         enhanced_img = enhancer.enhance(0.049)
12     return np.asarray(enhanced_img,dtype='uint8')
```

```
1  def enhance_image_sharpness(image_array,operation):
2      img = array_to_img(np.reshape(image_array,(48,48,1)))
3
4      enhancer = ImageEnhance.Sharpness(img)
5      if(operation == 'increase'):
6          enhanced_img = enhancer.enhance(2.35)
7      else:
8          enhanced_img = enhancer.enhance(0.049)
9      return np.asarray(enhanced_img,dtype='uint8')
```

```
1  from Data_Preprocessing import *
2
3  def Brightness_And_Sharpness_Augmented_data(x_train,y_train,data_set_len):
4      inc_brighteness_x = []
5      inc_brighteness_y = []
6      dec_brighteness_x = []
7      dec_brighteness_y = []
8      inc_sharpness_x = []
9      inc_sharpness_y = []
10     dec_sharpness_x = []
```

MODEL ARCHITECTURE:

Modification to VGGFace/VGG16 Architecture:

Normal VGG(Visual Geometry Group Architecture) Model Architecture :

No. of Convolutional Blocks = 5

No. of Convolutional Layer in each Block = 2

No. of Fully Connected (Dense Layer) = 1

No. of Dropout Layer = 1 (0.35)

CNN_Model.py

```
import keras
import tensorflow as tf
from keras.models import *
from keras.layers import *
from keras.utils import plot_model
import matplotlib.pyplot as plt
from IPython.display import display,Image
from keras.optimizers import Adam
from keras.callbacks import *

def CNN_Model_Initialize(n_classes,n_len = 1):
    #feature extraction
    model = Sequential()
    input_tensor = Input((48,48,1))
    out = input_tensor
    # modifing x -- that is modyfying the output tensor
    # No. of Colvolution blocks = 5
    for n_blocks in range(5):
        if n_blocks < 2:
            convl_num = 2 #no. of convoulutional Layer in each block = 2
        else :
            convl_num = 3
        #Building each convolution Layer
        for i in range(convl_num):
            out=Conv2D(32*2**min(3,n_blocks),kernel_size=3,activation='relu',padding='same',kernel_initializer='he_uniform')(out)
        # One Pooling Layer per block
        out = MaxPooling2D(pool_size=(2,2),strides=(2,2),padding='same')(out) # Stride is the number of pixels shifts over the input
        out = Dropout(0.35)(out)
    out = Flatten()(out)
    #fully-Connected Layer
    hidden_layer_dim = 2**10
    out = Dense(hidden_layer_dim,activation='relu')(out)
    out = Dropout(0.35)(out)
    out = [Dense(n_classes,activation='softmax')(out) for i in range(n_len)]
    model = Model(inputs=input_tensor,outputs=out)
    return model

def CNN_model_visualize(model):
    plot_model(model,to_file='CNN_Model_Plot.png',show_shapes=True)
    display(Image('CNN_Model_Plot.png'))
```

```python
def CNN_model_Compile_and_Train(model,X_train,Y_train,train_num,batch_size,epoch):
    # Patience= 4 ->The no. no of epochs with no improvement - metric is "loss"
    # CsvLogger -> streams the epoch Results to a csv file
    # Adam Optimizer --> Learning Rate Automatically Set
    model.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.001*(0.1**(float(train_num-1))),amsgrad=True),metrics=['accuracy'])
    call_backs=[CSVLogger("CNN_Model_Epochs.csv"),ModelCheckpoint('CNN_Model.h5',save_best_only=True)]
    history=model.fit(X_train,Y_train,batch_size=batch_size,epochs=epoch,verbose=1,validation_split=0.20,callbacks=call_backs,shuffle=True)
    return model,history
```

1. Importing the modules and the dataset:

Real Time Emotion Detection from Facial Expression using CNN

```python
#https://www.kaggle.com/richadey/data-preprocessing-and-augmented-data-preparation?scriptVersionId=44713517
```

```python
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from Utils_funX import *
from Data_Preprocessing import *
from CNN_Model import *
from keras.layers import *
```

Importing the dataset

```python
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
```

2.Exploratory Data Analysis:

2.1. Exploratory Data Analysis on the Training Data

Exploratory Data Analysis : Training Data

```python
print("\nThe shape of the Traning samples = {} \n".format(train_data.shape))
train_data
```

```
The shape of the Traning samples = (28709, 2)
```

	emotion	pixels
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1	0	151 150 147 155 148 133 111 140 170 174 182 15...
2	2	231 212 156 164 174 138 161 173 182 200 106 38...
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...
...	...	...
28704	2	84 85 85 85 85 85 85 85 86 86 86 87 86 86 91 9...
28705	0	114 112 113 113 111 111 112 113 115 113 114 11...
28706	4	74 81 87 89 95 100 98 93 105 120 127 133 146 1...
28707	0	222 227 203 90 86 90 84 77 94 87 99 119 134 14...
28708	4	195 199 205 206 205 203 206 209 208 210 212 21...

28709 rows × 2 columns

```
# Description of the train dataset
train_data.describe()
```

	emotion
count	28709.000000
mean	3.317427
std	1.876632
min	0.000000
25%	2.000000
50%	3.000000
75%	5.000000
max	6.000000

```
#information of the training data set
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28709 entries, 0 to 28708
Data columns (total 2 columns):
emotion    28709 non-null int64
pixels     28709 non-null object
dtypes: int64(1), object(1)
memory usage: 448.7+ KB
```

```
# Looking For the no. of null values in the Training dataset
train_data[train_data.columns].isna().sum()
```

```
emotion    0
pixels     0
dtype: int64
```

2.2. Exploratory Data Analysis On the Testing Data

Exploratory Data Analysis : Testing Data

+ Code

```
[ ]  print("\nThe shape of the Testing samples = {} \n".format(test_data.shape))
     test_data
```

```
The shape of the Testing samples = (7178, 1)
```

	pixels
0	254 254 254 254 254 249 255 160 2 58 53 70 77 ...
1	156 184 198 202 204 207 210 212 213 214 215 21...
2	69 118 61 60 96 121 103 87 103 88 70 90 115 12...
3	205 203 236 157 83 158 120 116 94 86 155 180 2...
4	87 79 74 66 74 96 77 80 80 84 83 89 102 91 84 ...
...	...
7173	50 36 17 22 23 29 33 39 34 37 37 37 39 43 48 5...
7174	178 174 172 173 181 188 191 194 196 199 200 20...
7175	17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...
7176	30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...
7177	19 13 14 12 13 16 21 33 50 57 71 84 97 108 122...

7178 rows × 1 columns

```
1 test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7178 entries, 0 to 7177
Data columns (total 1 columns):
pixels    7178 non-null object
dtypes: object(1)
memory usage: 56.2+ KB
```

```
1 # Looking For the no. of null values in the Testing dataset
2 test_data[test_data.columns].isnull().sum()
```

```
pixels    0
dtype: int64
```

```
1 # Looking for the Duplicate rows in the Test Dataset
2 test_data[test_data.duplicated()]
```

	pixels
696	67 82 94 112 99 88 64 76 137 157 164 184 198 2...
806	212 212 212 211 209 215 187 121 133 144 150 15...
953	251 251 251 249 255 231 72 7 3 9 17 24 51 70 1...
1101	42 41 47 48 46 54 59 62 73 82 97 100 97 103 10...
1433	49 38 30 43 46 38 79 116 140 150 157 166 171 1...
...	...
6851	42 41 47 48 46 54 59 62 73 82 97 100 97 103 10...
6898	214 215 213 210 93 35 43 96 118 118 118 119 11...
6901	133 183 181 194 181 179 207 205 211 194 195 21...
6927	20 18 18 18 19 19 20 20 21 19 19 20 19 19 19 1...

3. Cleaning the data

Cleaning the Data

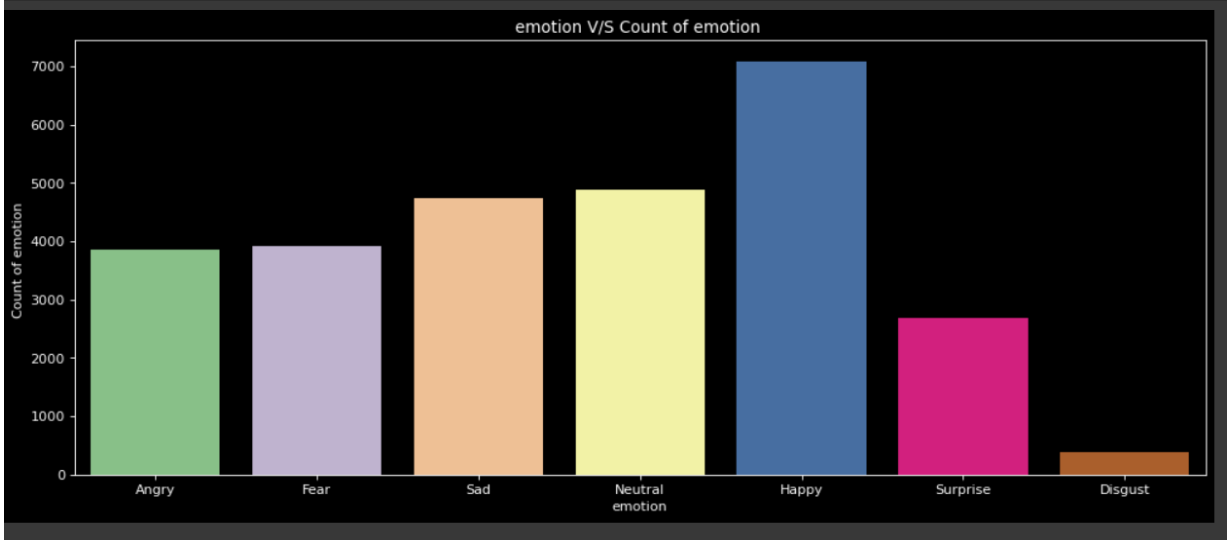
```
1 #Since we have no Null values in the dataset--> we only remov the Duplicates
2 train_data = remove_duplicates(train_data)
3 test_data = remove_duplicates(test_data)
4 print("\nThe shape of the Traning samples after Data Preprocessing is {} \n".format(train_data.shape))
5 print("\nThe shape of the Testing samples after after Data Preprocessing is {} \n".format(test_data.shape))
```

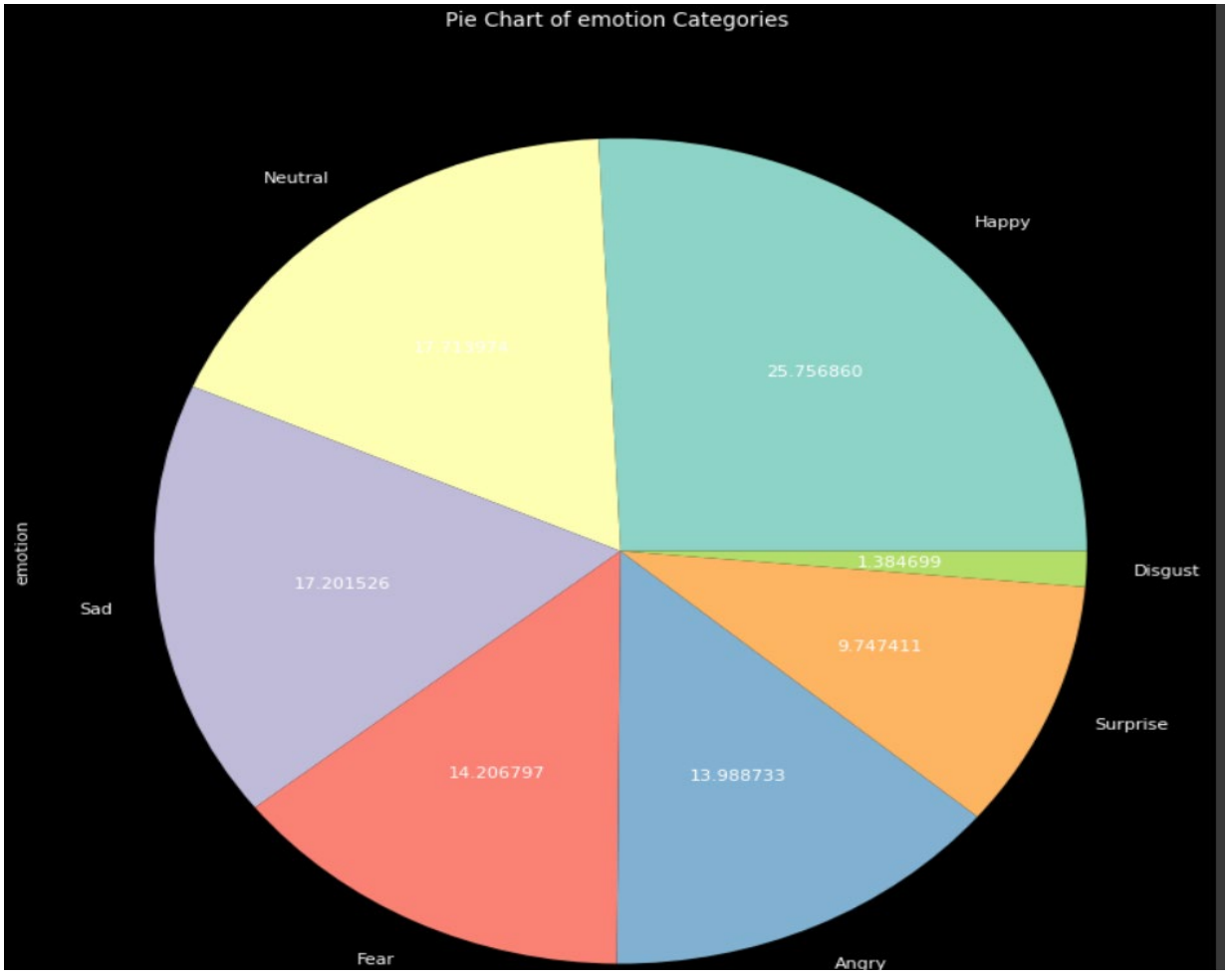
```
The shape of the Traning samples after Data Preprocessing is (27515, 2)

The shape of the Testing samples after after Data Preprocessing is (7092, 1)
```

4. Visualizing the Data

```
dictionary = {
    0:"Angry",
    1:"Disgust",
    2:"Fear",
    3:"Happy",
    4:"Sad",
    5:"Surprise",
    6:"Neutral" }
temp_dataset=train_data["emotion"].replace(dictionary)
temp_dataset=pd.DataFrame(temp_dataset)
Bar_Plots_For_Features(temp_dataset,"emotion")
```





5. Train and Test Data Preparation

```
Train and Test Data Preparation

1 x_train,y_train,x_test,y_test = Data_Preparation(train_data,test_data)

+ Code  + Text

[ ]  1 print("\nThe train and test data Shapes are :",x_train.shape,y_train.shape,x_test.shape,y_test.shape)

The train and test data Shapes are : (27515, 2304) (27515,) (7092, 2304) (0,)
```

6. Normalization of Test and Train Data

```
Normalization

[ ]  1 x_train,x_test=Data_Normalization(x_train,x_test)

[ ]  1 x_train,y_train,x_test,y_test

(array([[0.01138692, 0.01301362, 0.01333896, ..., 0.01724305, 0.01773106,
         0.01333896],
        [0.02358511, 0.02342892, 0.02296034, ..., 0.03014521, 0.02858328,
         0.02873947],
        [0.0271399 , 0.02490762, 0.01832825, ..., 0.01033901, 0.01292376,
         0.01785829],
        ...,
        [0.0102539 , 0.01122386, 0.01205526, ..., 0.02605044, 0.02591187,
         0.02591187],
        [0.03209718, 0.03282008, 0.02935012, ..., 0.01966313, 0.01966313,
         0.01937397],
        [0.02237966, 0.02283873, 0.02352734, ..., 0.00068861, 0.00172151,
         0.00436117]]),
 array([0, 0, 2, ..., 4, 0, 4]),
 array([[0.03202291, 0.03202291, 0.03202291, ..., 0.00529513, 0.0162636 ,
         0.0226934 ],
        [0.01826705, 0.02154575, 0.0231851 , ..., 0.02014059, 0.01955511,
         0.01885253],
```

7. One Hot Encoding

```
1 height = 48
2 width =48
3 x_train = x_train.reshape(x_train.shape[0],width,height,)
4 x_test = x_test.reshape(x_test.shape[0],width,height,)

1 x_train.shape

(27515, 48, 48)

1 from keras.utils import to_categorical
2 y_train = to_categorical(y_train,7)
3 print(y_train.shape)
4 y_train

(27515, 7)
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]], dtype=float32)

1 y_test = to_categorical(y_test,7)
2 print(y_test.shape)
3 y_test

(0, 7)
array([], shape=(0, 7), dtype=float32)
```

8. Data Augmentation

8.1. Brightness Augmented Data Demonstration

Data Augmentation

```
#!pip3 install import_ipynb
```

```
import import_ipynb
from Brightness_And_Sharpness_Augmented_data2 import *
```

```
importing Jupyter notebook from Brightness_And_Sharpness_Augmented_data2.ipynb
```

```
# Demonstration of Brightness Augmented Data

img1 = enhance_image_brighness(x_train[0],'increase')
img2 = enhance_image_brighness(x_train[0],'decrease')

plt.title(Decode_Y_Val(y_train[0]))
plt.imshow(x_train[0])
plt.show()

plt.title(Decode_Y_Val(y_train[0]))
plt.imshow(img1)
plt.show()

plt.title(Decode_Y_Val(y_train[0]))
plt.imshow(img2)
plt.show()
```

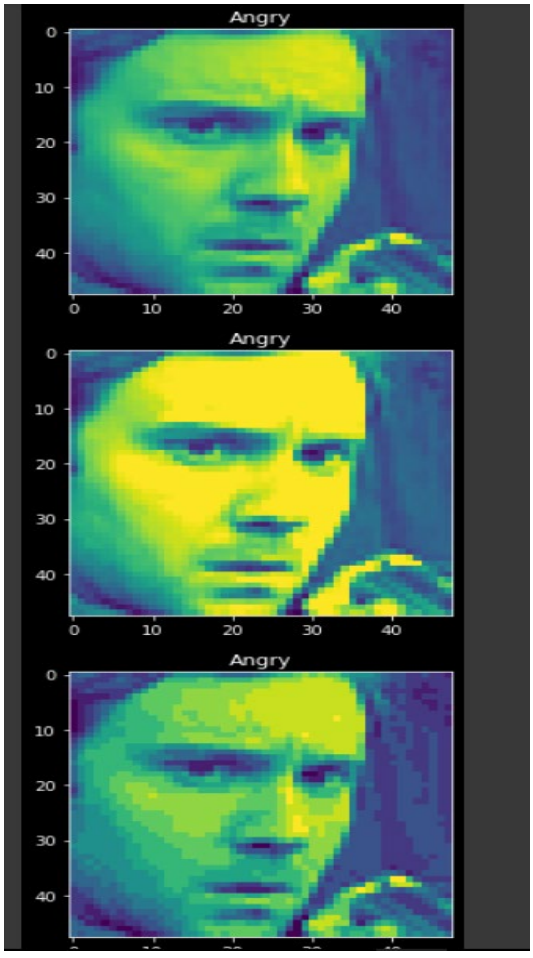


Fig 1. Original Image

Fig 2. Enhanced Brightness

Fig 3. Decreased Brightness

8.1. Brightness Augmented Data Demonstration

```
img1 = enhance_image_sharpness(x_train[0],'increase')
img2 = enhance_image_sharpness(x_train[0],'decrease')

plt.title(Decode_Y_Val(y_train[0]))
plt.imshow(x_train[0])
plt.show()

plt.title(Decode_Y_Val(y_train[0]))
plt.imshow(img1)
plt.show()

plt.title(Decode_Y_Val(y_train[0]))
plt.imshow(img2)
plt.show()
```

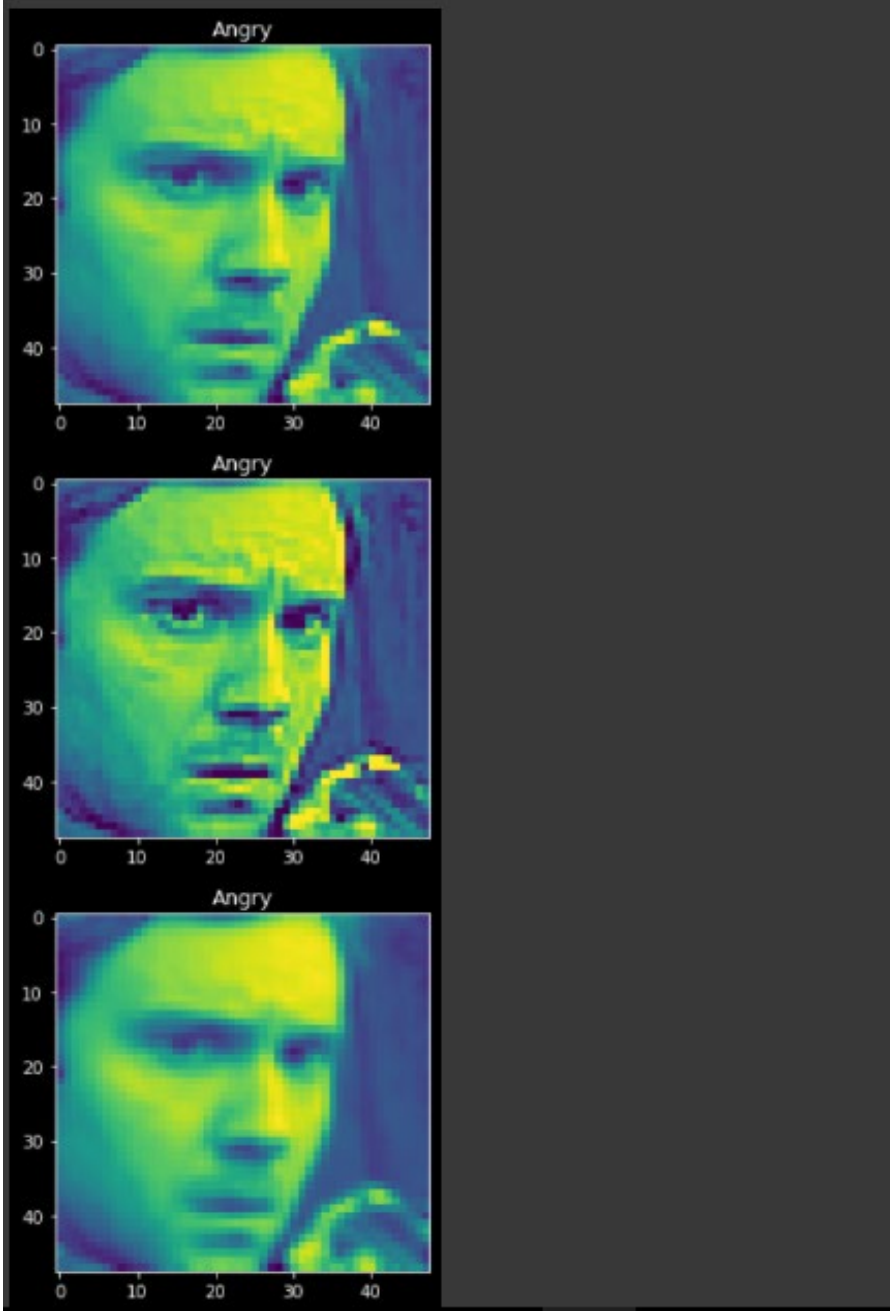


Fig 1. Original Image

Fig 2. Enhanced Sharpness

Fig 3. Decreased Sharpness

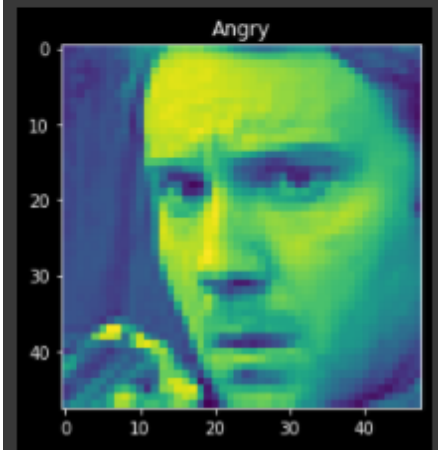
8.3. Vertical Augmentation(Mirror Image)

```
# adding two type of data 1 -> vertically flipped and horizontally fliped
# therefore After augmentation

x_train1,y_train1=Brightness_And_Sharpness_Augmented_data(x_train,y_train,len(x_train))
x_train2,y_train2=Data_Augmentation(x_train,y_train)

x_train = np.concatenate((x_train1,x_train2))
y_train = np.concatenate((y_train1,y_train2))
```

```
Five Sample of image after vertical flip :
```



9. Train-Test Split

```
from sklearn.model_selection import train_test_split
x_train,x_test1,y_train,y_test1 = train_test_split(x_train,y_train,test_size=0.20,random_state = 42)
```

10. Initializing the Model

Initialize some values

```
num_features = 64
n_classes = 7
batch_size = 64
epochs = 100
width = 48
height = 48
```

Training the Model

```
[ ]  from CNN_Model import *
```

```
[ ]  model= CNN_Model_Initialize(height,width,n_classes)
```

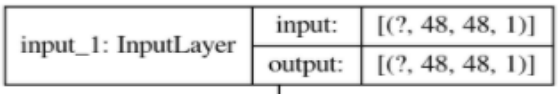
11. Model Summary:

```
model.summary()
```

```
=================================================================
Total params: 4,736,359
Trainable params: 4,736,359
Non-trainable params: 0
```

12. Visualizing the Model: Saved as **CNN_Model_Plot.png**

```
CNN_model_visualize(model)
```



13. Compiling and Training Our Model

```
from keras import *
```

```
#First Phase of Training
model,history=CNN_model_Compile_and_Train(model,x_train,y_train,1,200)
model.save('CNN_Model_Final.h5', include_optimizer=False)
```

```
Epoch 200/200
1651/1651 [==============================] - 28s 17ms/step - loss: 0.3440 - accuracy: 0.8839 - val_loss: 0.3261 - val_accuracy: 0.8995
```

At the 200th epoch we get:

Training Loss = 0.344 Training Accuracy = 0.88

Validation Loss = 0.32 Validation Accuracy = 0.899

We save the model as **CNN_Model_Final_v1.h5** and the model history in **CNN_Model_Epochs.csv**

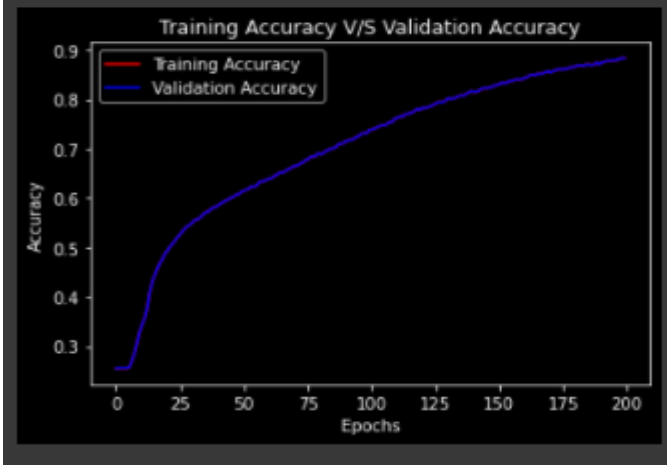
14. Plotting the Training Loss Versus Validation Loss

```
# Plotting the Training loss v/s validation Loss
plot_training_loss_vs_validation_loss(history)
```

We can't see the red line because it is behind the blue line, since we are plotting in range (0-200). We can see the variations when we take smaller range.

15. Plotting the Training Accuracy Versus Validation Accuracy

```
1 # Plotting the Training Accuracy v/s validation accuracy
2 plot_training_accuracy_vs_validation_loss(history)
```



16. TESTING AND EVALUATING THE PERFORMANCE OF THE MODEL ON STATIC DATA

16.1. Testing on Static Data

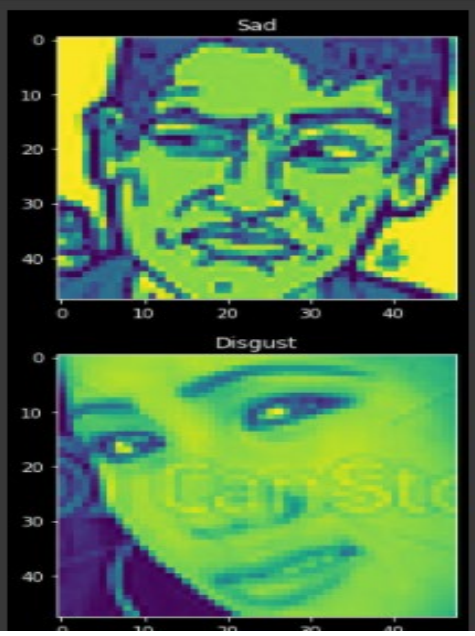
TESTING AND EVALUATION OF THE PERFORMANCE THE MODEL

```
[ ]  1 #First set of Test Samples(Not HAving test Results)
     2 x_test.shape,y_test.shape
```

```
((7092, 48, 48), (0, 7))
```

```
[ ]  1 y_test = model.predict(x_test,batch_size=100)
```

```
[ ]  1 #First set of Test Samples Without Predicted Value
     2 import matplotlib.pyplot  as plt
     3 from Utils_funX import *
     4
     5 for i in range(50):
     6     plt.title(Decode_Y_Val(y_test[i]))
     7     plt.imshow(x_test[i])
     8     plt.show()
```



16.2. Evaluation of the Metrics:

```
1 results = model.evaluate(x_test1,y_test1,batch_size=100,verbose=0)
2
3 print("Test Loss And Test Accuracy :\n",results)
```

```
Test Loss And Test Accuracy :
 [0.3362025320529938, 0.8963898420333862]
```

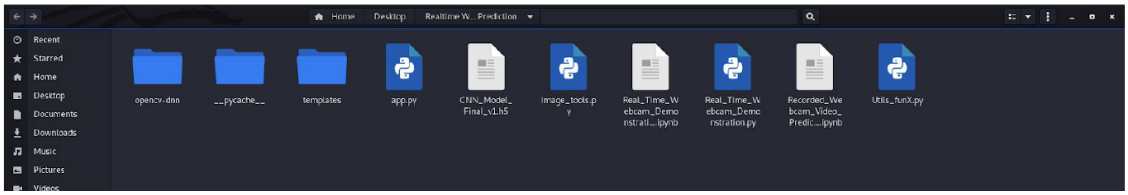
Test Loss = 0.3362

Test Accuracy = 0.8963

17. TESTING AND EVALUATING THE PERFORMANCE OF THE MODEL ON REALTIME DATA

1.1. Realtime Webcam Prediction Version 1(Testing on Local Host)

1. Unzip the zip file "Realtime Webcam Prediction.zip" and open the folder “Realtime Webcam Prediction”



2. Open the terminal in the Project directory
3. In terminal:
    3.1. export FLASK_ENV=development
    3.2. python3 app.py or flask run

4. Go to http://127.0.0.1:5000/ in your browser(Chrome preferably)

Realtime_Webcam_Demonstration.py

```
import cv2
from imutils.video import VideoStream
import numpy as np
import pandas as pd
from keras import *
from keras.preprocessing import image
from Utils_funX import *
import image_tools as lit
from PIL import Image,ImageEnhance

prototxt_path = './opencv-dnn/deploy.prototxt'
caffemodel_path ='./opencv-dnn/weights.caffemodel'
net = cv2.dnn.readNetFromCaffe(prototxt_path,caffemodel_path)

model = models.load_model('CNN_Model_Final_v2.h5')
#model.summary()


#facial_haarcascade_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
capture = cv2.VideoCapture(0)

ex = 0
while(True):
    #Capturing the frame
    ret , facial_img = capture.read()
    if not ret:
        continue
    (h,w) = facial_img.shape[:2]
    gray_scale_img = np.array(lit.rgb2gray_approx(facial_img),dtype = 'uint8')
    blob = cv2.dnn.blobFromImage(cv2.resize(facial_img,(1000,1000)),1.0,(300,300),(104.0, 177.0, 123.0))
    net.setInput(blob)
    detections = net.forward()

    for i in range(0,detections.shape[2]):
        #Bounding Box
        confidence = detections[0,0,i,2]
        box = detections[0,0,i,3:7]*np.array([w,h,w,h])

        (s_x,s_y,e_x,e_y) = box.astype("int")

        if(confidence > 0.9 ):
```

```
            cv2.rectangle( facial_img , (s_x,s_y) , (e_x,e_y) , color = (0,0,0) , thickness = 1 )
            #Cropping Out the Region of interest to feed to our trained model
            gray_cropped = cv2.resize(gray_scale_img[s_y-20:e_y+20,s_x-10:e_x+10],(48,48))

            #getting the image pixels
            face_img_pixels1 = image.img_to_array(gray_cropped)

            face_img_pixels = np.expand_dims(face_img_pixels1,axis=0)
            face_img_pixels = face_img_pixels.astype(float)

            face_img_pixels /= 255 # Normalization

            predicted_label = model.predict(face_img_pixels)
            predicted_emotion = Decode_Y_Val(predicted_label)

            #predicted_emotion = "I am Always Sad"
            predicted_emotion = predicted_emotion.upper()
            cv2.putText(facial_img,predicted_emotion,(int(s_x),int(s_y)-10),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2,cv2.LINE_AA)
           ]
            Final_Result = cv2.resize(facial_img,(1500,800))
            cv2.imshow('Real Time Emotion Detection',Final_Result)

        if cv2.waitKey(10) == ord('q'):
            ex = 1
            break

    if ex == 1 :
        break
capture.release()
cv2.destroyAllWindows()
```

5. The WebCam will open.

6. Press ‘q’ to quit
8. To see the frame wise prediction of the system run “” in Jupyter Notebook

1.2. Realtime Webcam Prediction Version 2(Testing on Heroku)

Steps:

0. Unzip the file ‘Realtime Webcam Prediction Version2.zip’

1. Created a virtual environment in my project directory.

```
mishadey@kali:~/Desktop/app$ python3 -m venv venv/
```

2. Activated the virtual environment

```
mishadey@kali:~/Desktop/app$ source venv/bin/activate
```

3. Installed the required libraries using pip/pip3 command

```
(venv) mishadey@kali:~/Desktop/app$ pip3 install flask gunicorn opencv-python tensorflow-cpu numba flask_socketio flask_cors pillow gevent-websocket
```

4. Create **app.py**

```python
import io
import base64
from PIL import Image
from io import StringIO
import cv2
import numpy as np
import tensorflow
from tensorflow.keras import *
from Utils_funX import *
from tensorflow.keras.preprocessing import image
from flask import Flask, render_template, Response
from flask_socketio import SocketIO, emit
from flask_cors import CORS, cross_origin

prototxt_path = 'deploy.prototxt'
caffemodel_path ='weights.caffemodel'
net = cv2.dnn.readNetFromCaffe(prototxt_path,caffemodel_path)

app = Flask(__name__)
socketio = SocketIO(app)

def getModel():
    global classifier
    classifier = tensorflow.keras.models.load_model('CNN_Model_Final_v1.h5')
getModel()

cors = CORS(app)
#app.config['CORS_HEADERS'] = 'Content-Type'
@app.route('/', methods=['POST', 'GET'])
@cross_origin()

def index():
    return render_template('index.html')

@socketio.on('image')
def image(data_image):
    sbuf = StringIO()
    sbuf.write(data_image)
    # decode and convert into image
    b = io.BytesIO(base64.b64decode(data_image))
    facial_img = Image.open(b)
```

```
    (h,w) = np.float32(facial_img).shape[:2]

    gray_scale_img = np.array(cv2.cvtColor(np.float32(facial_img), cv2.COLOR_BGR2GRAY),dtype = 'uint8')

    blob = cv2.dnn.blobFromImage(cv2.resize(np.float32(facial_img),(1000,1000)),1.0,(300,300),(104.0, 177.0, 123.0))

    net.setInput(blob)
    detections = net.forward()

    for i in range(0,detections.shape[2]):

        confidence = detections[0,0,i,2]
        box = detections[0,0,i,3:7]*np.array([w,h,w,h])
        (s_x,s_y,e_x,e_y) = box.astype("int")

        if(confidence > 0.19 ):
            gray_cropped = cv2.resize(gray_scale_img[s_y-20:e_y+20,s_x-10:e_x+10],(48,48))

            face_img_pixels1 = tensorflow.keras.preprocessing.image.img_to_array(gray_cropped)

            face_img_pixels = np.expand_dims(face_img_pixels1,axis=0)
            face_img_pixels = face_img_pixels.astype(float)

            face_img_pixels /= 255

            predicted_label = classifier.predict(face_img_pixels)
            predicted_emotion = Decode_Y_Val(predicted_label)

            predicted_emotion = predicted_emotion.upper()

            emit('response_back',predicted_emotion)

if __name__ == '__main__':
    socketio.run(app, host='0.0.0.0',debug=True)
```

5. Create a requirements.txt with all the modules in required

```
(venv) mishadey@kali:~/Desktop/app$ pip3 freeze > requirements.txt
```

Contents of requirements.txt

```
Flask==1.1.2
Flask-Cors==3.0.9
Flask-SocketIO==4.3.1
gevent-websocket==0.10.1
gunicorn==20.0.4
h5py==2.10.0
idna==2.10
numba==0.51.2
numpy==1.18.5
opencv-contrib-python-headless
Pillow==8.0.1
tensorflow-cpu==2.3.1
```

6. Create a file Procfile

Content:

```
web: gunicorn app:app
```

7. Create a new app in Heroku named **'real-time-emotion-detection'**

8. Log in to Heroku

```
(venv) mishadey@kali:~/Desktop/app$ heroku login
```

9. Initializing a git repository with 'git init'

```
(venv) mishadey@kali:~/Desktop/app$ git init .
Reinitialized existing Git repository in /home/mishadey/Desktop/app/.git/
```

10. Add all necessary files to the repository

```
(venv) mishadey@kali:~/Desktop/app$ git add app.py  CNN_Model_Final_v1.h5  deploy.prototxt  image_tools.py  Procfile  Real_Time_Webcam_Demonstration.py  requirements.txt  runtime.txt  templates
Utils_funX.py  weights.caffemodel
```

11. Commit the changes to repository

```
(venv) mishadey@kali:~/Desktop/app$ git commit -m "first"
```

12. Add the repository to the remote repo.

```
(venv) mishadey@kali:~/Desktop/app$ heroku git:remote -a real-time-emotion-detection
set git remote heroku to https://git.heroku.com/real-time-emotion-detection.git
```

13. Pushing the project to Heroku

```
(venv) mishadey@kali:~/Desktop/app$ git push heroku master
Enumerating objects: 55, done.
Counting objects: 100% (55/55), done.
Delta compression using up to 8 threads
Compressing objects: 100% (47/47), done.
Writing objects: 100% (55/55), 21.64 MiB | 1.38 MiB/s, done.
Total 55 (delta 20), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Python app detected
remote: -----> Installing python-3.8.6
remote: -----> Installing pip 20.1.1, setuptools 47.1.1 and wheel 0.34.2
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
```

```
remote: -----> Compressing...
remote:        Done: 327.9M
remote: -----> Launching...
remote:  !     Warning: Your slug size (327 MB) exceeds our soft limit (300 MB) which may affect boot time.
remote:        Released v3
remote:        https://real-time-emotion-detection.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/real-time-emotion-detection.git
 * [new branch]      master -> master
```

14. Now for the openCV modules to work fine we instal the Heroku-buildpack-multi

```
(venv) mishadey@kali:~/Desktop/app$ heroku config:add BUILDPACK_URL=https://github.com/ddollar/heroku-buildpack-multi.git --app real-time-emotion-detection
Setting BUILDPACK_URL and restarting ⬢ real-time-emotion-detection... done, v4
BUILDPACK_URL: https://github.com/ddollar/heroku-buildpack-multi.git
```

Heroku Link for Realtime Emotion Detection Model:

https://real-time-emotion-detection.herokuapp.com/

Suggestions:

1. For Faster Prediction,GPUs are Required(Since Opencv DNN is being used)
2. For Prediction at farther distance, Better resolution Cameras required
3. Model accuracy ~ 89%