VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

A PROJECT REPORT ON

"SENTIMENT ANALYSIS USING FACE RECOGNITION"

Submitted in Partial fulfillment of the Requirements for the Degree of

Master of Computer Applications

Submitted by

AKHIL K K (1CR20MC005)

Under the Guidance of,
Dr.Gnaneswari G
Assistant Professor
Dept. of MCA

DEPARTMENT OF MCA

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

DEPARTMENT OF MCA

CERTIFICATE

Certified that the project work entitled **"SENTIMENT ANALYSIS USING FACIAL RECOGNITION"** carried out by **AKHIL K K** USN **1CR20MC005,** Bonafide student of CMR Institute of Technology, in partial fulfillment for the award of **Master of Computer Applications** of the Visveswaraya Technological University, Belgaum during the year 2021-2022. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Dr. Gnaneswari G

Assistant Professor

Dept. of MCA,

CMRIT

Ms. Gomathi T

Professor & Head Dept. of MCA,

CMRIT

DECLARATION

I, the student of MCA, CMR Institute of Technology, Bangalore declare that the work entitled "**SENTIMENT ANALYSIS USING FACIAL RECOGNITION**" has been successfully completed under the guidance of Dr. Gnaneswari G, Assistant Professor, Department of MCA, CMR Institute of Technology, Bangalore. This dissertation work is submitted in partial fulfillment of the requirements for the award of Degree of Master of Computer Applications during the academic year 2021 - 2022. Further the matter embodied in the project report has not been submitted previously by anybody for the award of any degree or diploma to any university.

Place:

Date:

AKHIL K K (1CR20MC005) _____

ACKNOWLEDGEMENT

At the various stages in making the mini project, a number of people have given me invaluable comment on the manuscript. I take this opportunity to express my deepest gratitude and appreciation to all those who helped me directly or indirectly towards the successful completion of this project.

I would like to thank **Principal Dr. Sanjay Jain, CMR Institute of Technology** for his support throughout this project.

I express my whole hearted gratitude to **Prof. Gomati T,** who is our respectable **Head of Dept, MCA.** I wish to acknowledge for her valuable help and encouragement.

In this regard we owe a heartfelt gratitude to my guide **Dr. Gnaneswari G, Assistant Professor, Department of MCA,** for her timely advice on the mini project and regular assistance throughout the project work. I would also like to thank the staff members of Department of MCA for their corporation.

ABSTRACT

The sentimental analysis is a process of extracting human feelings from data . It is an application of natural language processing , computational linguistics, text analysis. Its basic idea is to classify human emotions in different moods such as happy, sad, neutral, etc. It uses the pattern of movement of both the lips and the eye shape that it takes during different feelings of human. It has a huge variety of applications because of its ability to extract insights from data sets and the social media. It will be using machine learning algoriths for the detection of emotions and it will be trained on the huge dataset with varying sample size. It will also use facial recognition to perform a specific analysis of a person after identifying their face. This will provide a separate report for each person on their emotional state. This report then will be used for the expression mining in different modern systems such as online streaming, video interviews, etc. This report will help to empower the existing tools to perform multi task and gives a lot of data to work with.

MINI PROJECT

REPORT

SENTIMENT ANALYSIS USING FACIAL RECOGNITION

DONE BY:

AKHIL K K (1CR20MC005)

CHIRLAM CHARLA MANJUNATH (1CR20MC019)

INDEX

INTRODUCTION

A Facial expression is the visible manifestation of the affective state, cognitive activity, intention, personality and psychopathology of a person and plays a communicative role in interpersonal relations. It have been studied for a long period of time and obtaining the progress recent decades. Though much progress has been made, recognizing facial expression with a high accuracy remains to be difficult due to the complexity and varieties of facial expressions.

Generally human beings can convey intentions and emotions through nonverbal ways such as gestures, facial expressions and involuntary languages. This system can be significantly useful, nonverbal way for people to communicate with each other. The important thing is how fluently the system detects or extracts the facial expression from image. The system is growing attention because this could be widely used in many fields like lie detection, medical assessment and human computer interface.

The system classifies facial expression of the same person into the basic emotions namely anger, disgust, fear, happiness, sadness and surprise. The main purpose of this system is to detect sentiment of the face using the CNN model.

In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the receptive field. The receptive fields of different neurons partially overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation. Convolutional networks were inspired by biological processes and are variations of multilayer perceptron designed to use minimal amounts of pre-processing.

REQUIREMENT ANALYSIS / SOFTWARE REQUIREMENT SPECIFICATION

JUPYTER NOTEBOOK

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

You can start running a notebook server from the command line using the following command:

```
jupyter notebook
```

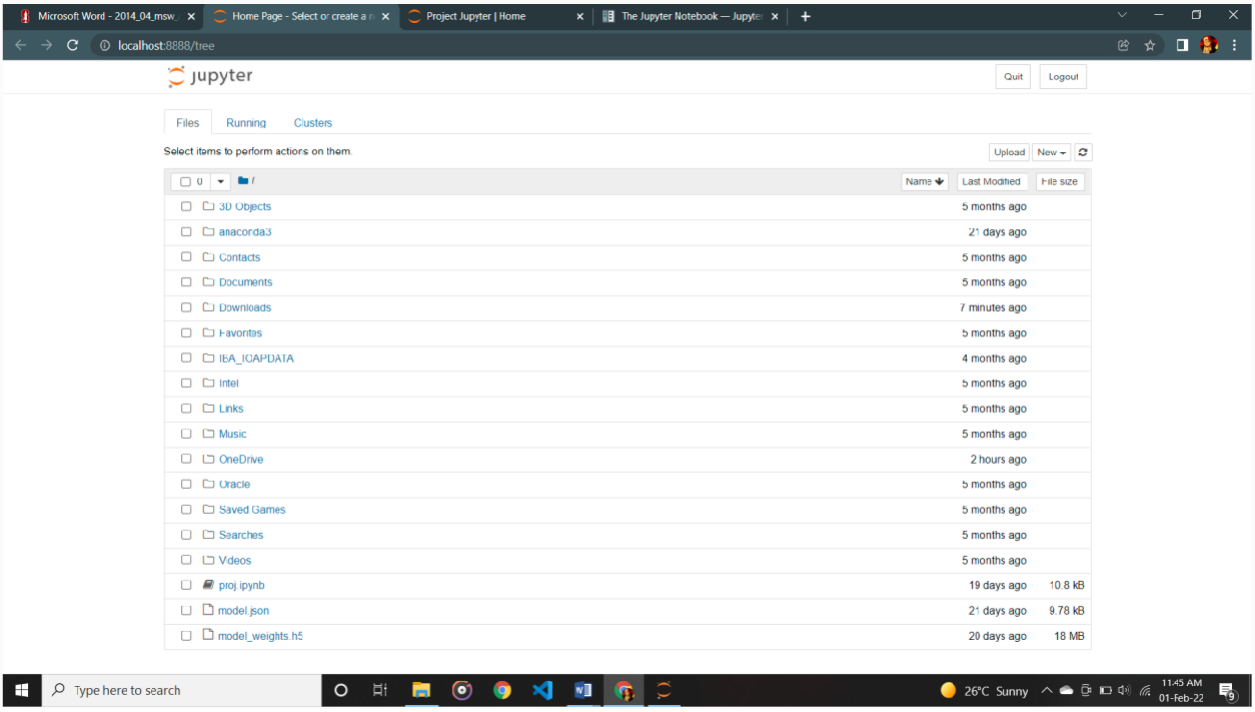
This will print some information about the notebook server in your console, and open a web browser to the URL of the web application (by default, http://127.0.0.1:8888).

The landing page of the Jupyter notebook web application, the **dashboard**, shows the notebooks currently available in the notebook directory (by default, the directory from which the notebook server was started).

You can create new notebooks from the dashboard with the `New Notebook` button, or open existing ones by clicking on their name. You can also drag and drop `.ipynb` notebooks and standard `.py` Python source code files into the notebook list area.

When starting a notebook server from the command line, you can also open a particular notebook directly, bypassing the dashboard, with `jupyter notebook my_notebook.ipynb`. The `.ipynb` extension is assumed if no extension is given.

When you are inside an open notebook, the *File | Open…* menu option will open the dashboard in a new browser tab, to allow you to open another notebook from the notebook directory or to create a new notebook.



GOOGLE COLAB

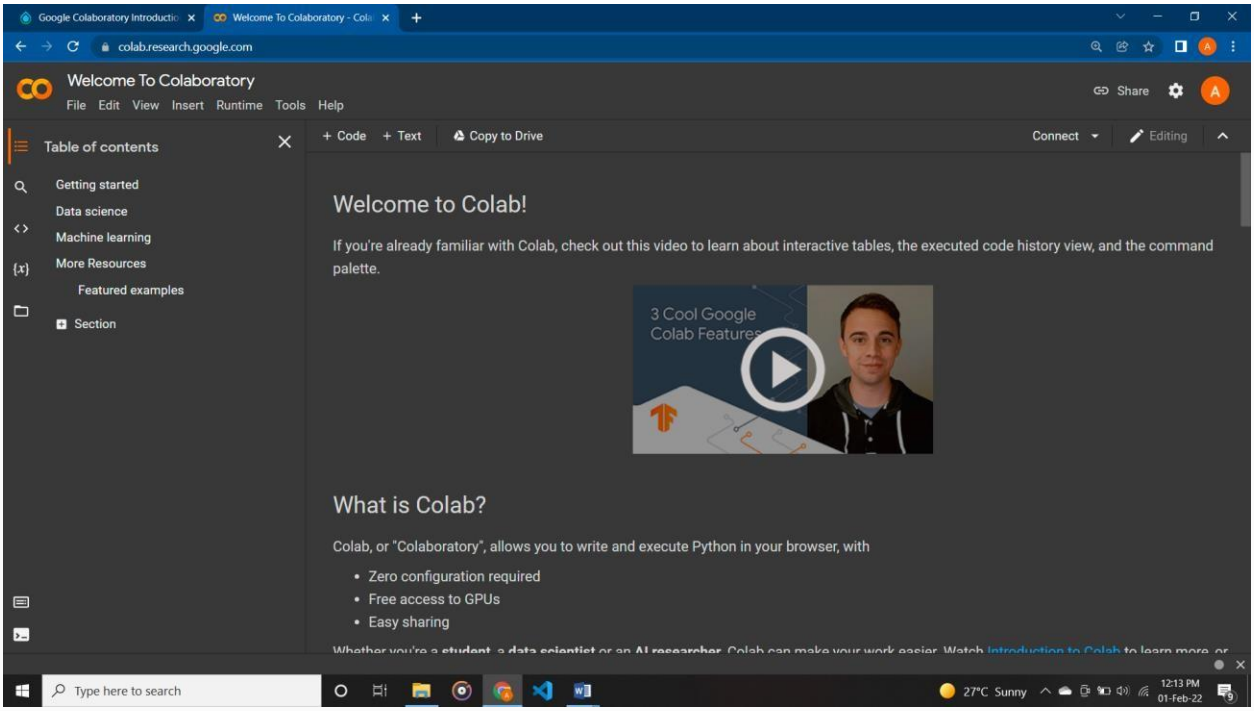
Colab is a free notebook environment that runs entirely in the cloud. It lets you and your team members edit documents, the way you work with Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook. Google is quite aggressive in AI research. Over many years, Google developed AI framework called **TensorFlow** and a development tool called **Colaboratory**. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as Google Colab or simply **Colab**.

Google Colaboratory, or Colab, is a cloud-based environment for writing documents with live code, visualizations, and narrative text. For those who are familiar with Jupyter notebooks, Colab notebooks are the same, including the .ipynb extension. Unlike Jupyter and Atom (our previous editor for code and reports), however, Colab requires no setup on your computer! It also provides a large amount of free computing power and easy document sharing.

A Colab notebook consists of **text cells**, **code cells**, and **outputs of code cells**.

1. Text cells are written in **Markdown**, a markup language we'll learn about in the next section. This means they can contain formatted text, images, HTML, LaTeX, and more.
2. Code cells are written in Python. We can also insert a system/terminal command by prefixing a line with !, like so:

   ```
   print("This is Python code.")
   !echo This is a system command.
   ```
3. Outputs of code cells appear below their corresponding cell. They can include text, graphics, and information about errors that occurred while executing the code.


Google Colaboratory Introductio
Welcome To Colaboratory - Cola
colab.research.google.com
Welcome To Colaboratory
File Edit View Insert Runtime Tools Help
Share
Table of contents
Getting started
Data science
Machine learning
More Resources
Featured examples
Section
+ Code + Text
Copy to Drive
Connect
Editing
Welcome to Colab!
If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.
3 Cool Google Colab Features
What is Colab?
Colab, or "Colaboratory", allows you to write and execute Python in your browser, with
Zero configuration required
Free access to GPUs
Easy sharing
Type here to search
27°C Sunny
12:13 PM
01-Feb-22

IMPLEMENTATION

WORKING

Sentiment Analysis using facial recognition is done using the following facial expressions :

- Angry
- Disgust
- Fear
- Happy
- Neutral
- Sad
- Surprise

CODE

//INSTALLING KAGGLE

```
!pip install -q Kaggle

!mkdir ~/.kaggle
!cp "/content/kaggle.json" ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

Now we will get the dataset on which our model will be trained and we will validate how good or bad our model has performed on that particular dataset, so

that we can improve the accuracy score. I have uploaded the dataset used by me on Kaggle so that anyone can access that.

// DOWNLOADING DATASET FROM KAGGLE

```
!kaggle datasets download -d aadityasinghal/facial-expression-dataset
!unzip /content/facial-expression-dataset.zip
pip install livelossplot==0.5.2
pip install python-utils
```

STEP 1 →

//IMPORTING THE REQUIRED PYTHON LIBRARIES LIKE numpy, seaborn, matplotlib, tensorflow.

```
import numpy as np
import seaborn as sns
import python_utils
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPoo
ling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model
from IPython.display import SVG, Image
import tensorflow as tf
from livelossplot.inputs.tf_keras import PlotLossesCallback
print("Tensorflow version:", tf.__version__)
```

In the above code we have imported different modules from keras which is a wrapping on tensorflow. We have imported particularly that modules from keras which will help in building a CNN model. At last line in above code we have printed the current version of tensorflow in the system.

STEP 2 →

Now we will generate training and testing (validation) batches so that our model could be trained and evaluated/validated on the test data. This is a very important step as without this we can't have a accurate model and also without training the model also doesn't know what it has to look for and also predict for.

```
img_size = 48
batch_size = 64
datagen_train = ImageDataGenerator(horizontal_flip=True)
train_generator = datagen_train.flow_from_directory("/content/test/test",
                                                    target_size=(img_size,
img_size),
                                                    color_mode="grayscale"
,
                                                    batch_size=batch_size,
                                                    class_mode='categorica
l',
                                                    shuffle=True)
datagen_validation = ImageDataGenerator(horizontal_flip=True)
validation_generator = datagen_validation.flow_from_directory("/content/tr
ain/train",
                                                    target_size=(img_size,
img_size),
                                                    color_mode="grayscale"
,
                                                    batch_size=batch_size,
                                                    class_mode='categorica
l',
                                                    shuffle=False)
```

In the above code we have defined the image size to 48 so each image will be reduced to a size of 48x48. After that we have defined the batch size equal to 64 which means that in each epoch ,i.e., when the model is ran through the training dataset at each cycle, the number of images that will be passed will be 64. Which means that model will take first 64 images for training through first epoch and will continue so on till all the epochs are completed.

After that we have used ImageDataGenerator from keras module which Generate batches of tensor image data with real-time data augmentation. Here in this we have kept horizontal flip True which means it will randomly flips input images horizontally.

Now we will perform the main and most important step ,i.e., generating train and test data images.

Firstly we will go for train data. Here we have used a function from datagen_train which is from ImageDataGenerator ,i.e., flow_from_directory which takes few parameters like the path of the dataset, target_size (size of the output image), color_mode (color of the output images, we have set gayscale which gives grey images), batch_size, class_mode (Determines the type of label arrays that are returned , we have specified categorical) and we have shuffle False.

Now we will do the above same for generating the test dataset images. The only change will be the path of the test folder.

STEP 3 →

As of now we have imported libraries, got dataset, and created train and test images. Now its time to move to another important step which is building the CNN Model.

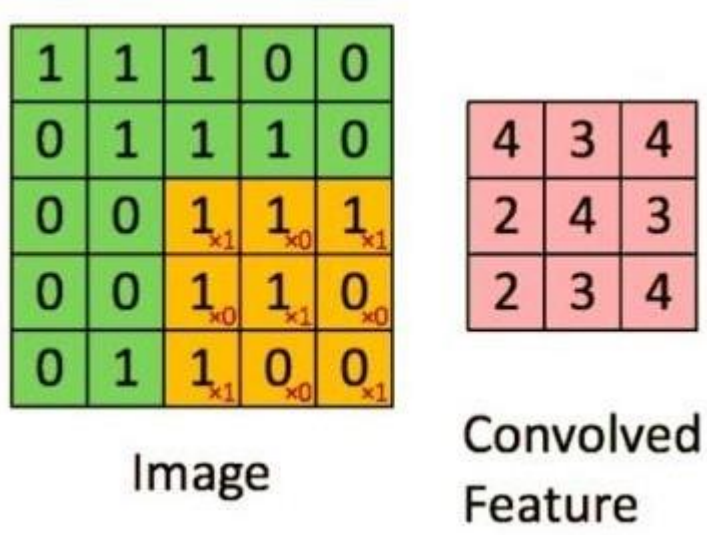
```
# Initialising the CNN
model = Sequential()
# 1 - Convolution
model.add(Conv2D(64,(3,3), padding='same', input_shape=(48, 48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# 2nd Convolution layer
model.add(Conv2D(128,(5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# 3rd Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# 4th Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# Flattening
model.add(Flatten())
# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(7, activation='softmax'))
opt = Adam(lr=0.0005)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['ac
curacy'])
model.summary()
```

In the above code we have created the basic structure of how our neural network will look. For this we have taken help from the keras module which is a great wrapper of the tensorflow library and helps to reduce our work. First of all we

have initialized the CNN model by using sequential() function. After that we have created the first four layer of the neural network which are Convolution Layer.

The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data. A convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. The multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel. The visualization for this is given below



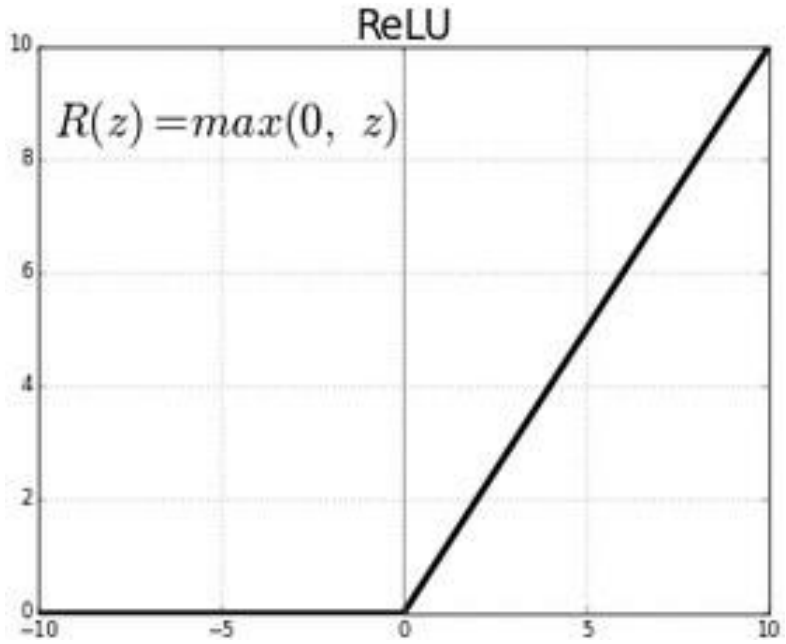
The convolutional layer can be added to our model by using model.add(). This takes few parameters like Conv2D which specifies that the layer to be added is convolutional layer. This method also takes two parameters , The first one the number of filters and the pooling. After that we apply Batch

Normalization which applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. After that activation function is added which does non linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input. And after that we have applied MaxPolling2D which take out only the maximum from a pool. And finally we add dropout which prevents the model from overfitting. And here we complete building a convolution layer.

After that we have added flatten layer which converts the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer.

Finally we have added to fully connected layers to the model. To create a fully connected layer first we add a Dense Layer which is a regular deeply connected neural network layer. It is most common and frequently used layer. After that we add Batch Normalization layer. And at last activation and dropout layer.

For all the layers till now we have kept the activation function ReLU because the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.



But after all the above layers we will finally add a Dense Layer layer with activation set as Softmax which turns numbers aka logits into probabilities that sum to one. Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes.

Thus In the output of above model we finally get probabilities in range 0 to 1 thus making easy for us to classify the expression.

After that we have used model.compile to compile the model. It takes some parameters like optimizer which optimize the input weights by comparing the prediction and the loss function. We have kept optimizer to Adam with learning rate (lr) specified. After that we have added loss to categorical_crossentropy and

we have kept metrices (which is used to evaluate the performance of model equal to) accuracy**.** Finally we have outputted model summary using model.summary() .

STEP 4 →

Now of as we have successfully built the model architecture , Now it's time to train the model and evaluate the results.

```
%%time
epochs = 5
steps_per_epoch = train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                              patience=2, min_lr=0.00001, mode='auto')
checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_accuracy',
                             save_weights_only=True, mode='max', verbose=1
)
callbacks = [PlotLossesCallback(), checkpoint, reduce_lr]
history = model.fit(
    x=train_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=epochs,
    validation_data = validation_generator,
    validation_steps = validation_steps,
    callbacks=[checkpoint]
)
```

The above code train our model on the training dataset and at the same time it validates on the test/validation dataset.

First of all We have set the number of epochs where One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE. Since one epoch is too big to feed to the computer at once we divide it in several smaller batches. As the number of epochs increases, more number of

times the weight are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve.

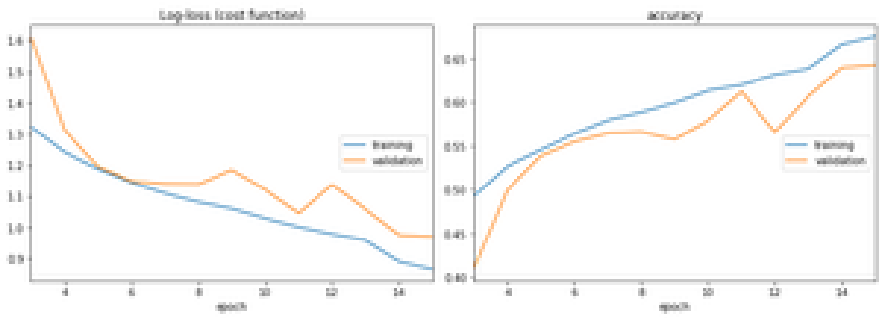
Important Note:- I have kept the number of epochs to 15 , you can increase that to gain more accuracy and have great results.

After that we have set the steps per epoch and validation steps equal to the the integer output of the division of the total number of images divided by the batch size.

After that we have set ReduceLROnPlateau that adjusts the learning rate when a plateau in model performance is detected, e.g. no change for a given number of training epochs. After that we have set the ModelCheckpoint which allows you to define where to checkpoint the model weights, how the file should named and under what circumstances to make a checkpoint of the model. After this we have set the Callbacks with PlotLossesCallback() which gives live report of the how the training is going on and we have added checkpoints and reduce_lr also.

Finally we have done model.fit() which starts the training and validation of

our model. Its takes some parameters as input like x -> training dataset which is training_generator after that steps per epoch, validation data , validation steps and callbacks. The output after running the above code is given below.



STEP 5 →

Now as we have got a trained model, Lets save the model in json format with its weights also saved.

```
model_json = model.to_json()
model.save_weights('model_weights.h5')
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

In the above code we have first converted model to json format. After that we have saved the weights of the model in .h5 format. After that we have opened a model.json file in write mode and we wrote the model which was converted into json format into this file. Now we have got two files named model.json and model_weights.h5 file for model and its weight respectively and which can be used anywhere to make predictions.

STEP 6 →

Now we will write a Python code for loading the model and weights and making Predictions.

```
from tensorflow.keras.models import model_from_json
class FacialExpressionModel(object):
    EMOTIONS_LIST = ["Angry", "Disgust",
                     "Fear", "Happy",
```

```
                  "Neutral", "Sad",
                  "Surprise"]
    def __init__(self, model_json_file, model_weights_file):
        # load model from JSON file
        with open(model_json_file, "r") as json_file:
            loaded_model_json  =  json_file.read()
            self.loaded_model = model_from_json(loaded_model_json)
        # load weights into the new model
        self.loaded_model.load_weights(model_weights_file)
        self.loaded_model.make_predict_function()
    def predict_emotion(self, img):
        self.preds = self.loaded_model.predict(img)
        return FacialExpressionModel.EMOTIONS_LIST[np.argmax(self.preds)]
```

In the above code we have first imported model_from_json function which helps us to import the model from a json file. After the we have written a python class in which it has first a list of emotions which our dataset contained. After that we have defined the init method which takes the model.json file and the model weights file which is in .h5 format. After that in this we are reading the json file and using the model_from_json function to load the model. After that we are loading the weights into the model.

After this in the Class we have defined a method named predict_emotion which gives the prediction of the image. First it uses .predict method to give prediction after that we are using the numpy argmax to get a integer number b/w 0–6 representing the corresponding emotion in the list. And finally we return that particular emotion name.

STEP 7 →

Now as we have mode the code to load the weights to model , Now we will *g*et the frames of the video and will perform the predictions on that.

```
Import cv2
facec = cv2.CascadeClassifier(r"C:\Users\akhil\anaconda3\Lib\site-
packages\cv2\data\haarcascade_frontalface_default.xml")
model = FacialExpressionModel("model.json", "model_weights.h5")
font = cv2.FONT_HERSHEY_SIMPLEX
class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)
    def __del__(self):
        self.video.release()
    # returns camera frames along with bounding boxes and predictions
    def get_frame(self):
        _, fr = self.video.read()
        gray_fr = cv2.cvtColor(fr, cv2.COLOR_BGR2GRAY)
        faces = facec.detectMultiScale(gray_fr, 1.3, 5)
        for (x, y, w, h) in faces:
            fc =  gray_fr[y:y+h,  x:x+w]
            roi = cv2.resize(fc, (48, 48))
            pred = model.predict_emotion(roi[np.newaxis, :, :,
np.newaxis])
            cv2.putText(fr, pred, (x, y), font, 1, (255, 255, 0), 2)
            cv2.rectangle(fr,(x,y),(x+w,y+h),(255,0,0),2)
        return fr
```

In the above code first of all we are importing the OpenCV Module. After this we are setting the CascadeClassifier with Haar Cascade Classifiers that is used to detect features by superimposing predefined patterns over face segments and are used as XML files. In our model.

After that we have called the model by passing the model.json file and weights file. After this we have set the font for the CV2. After that we have written a python class. In the class first of all we have declared init method which uses cv2 VideoCapture method to access the camera or the video file for which you want the predictions.

An Important Note :-

In the Code above we have passed 0 as argument in the VideoCapture. You can change the 0 to the path of the video file to make predictions on a video file. Here 0 means that the CV2 will get video from your PC's Webcam.

After that we have declared the destructor for the class which releases the video and stops the methods when you want to quite.

After that we have declared a method called get_frame which firsts read the video. After that we have used cv2.cvtColor() method which is used to convert an image from one color space to another. Here we have used cv2.COLOR_BGR2GRAY which converts the image to gray color as our model was trained on gray color images. After that we have used detectMultiScale() which detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles. After that we have have looped to the different coordinates of the image returned and resized the image using CV2.resize() function. Finally we have used model.predict_emotion to get the predicted emotion. After that we have put the text on the frame of the image which shows the predicted emotion and also we have put the rectangle box around the area where the face was detected. And Finally we have returned that frame along with the predicted box and text.

STEP 8 →

As of now we have mode all important functions , Now let us make the function for calling the above code and showing the output video.

```
def gen(camera):
    while True:
        frame = camera.get_frame()
        cv2.imshow('Facial Expression Recognization',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()
```

In the code above we have declared a python function named gen which takes the camera as parameter. In this we run a while loop for True , Continuously. And In this first we call get_frame function of VideoCamera**.** After that we use imshow method of CV2. to show the video as output. After this we have written code to stop the code. We have added if condition that if key 'q' is presses then loop will be break and the output screen will be destroyed using destroyAllWindows() function.

STEP 9 →

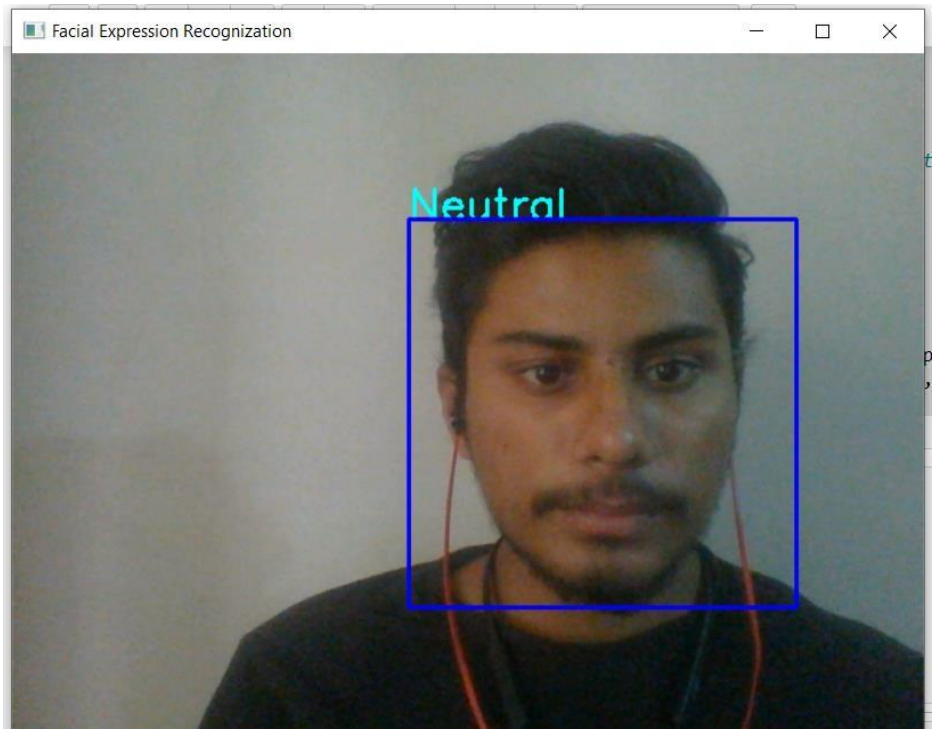
Now we will write a final line of code which will run all the code above by calling the gen function.

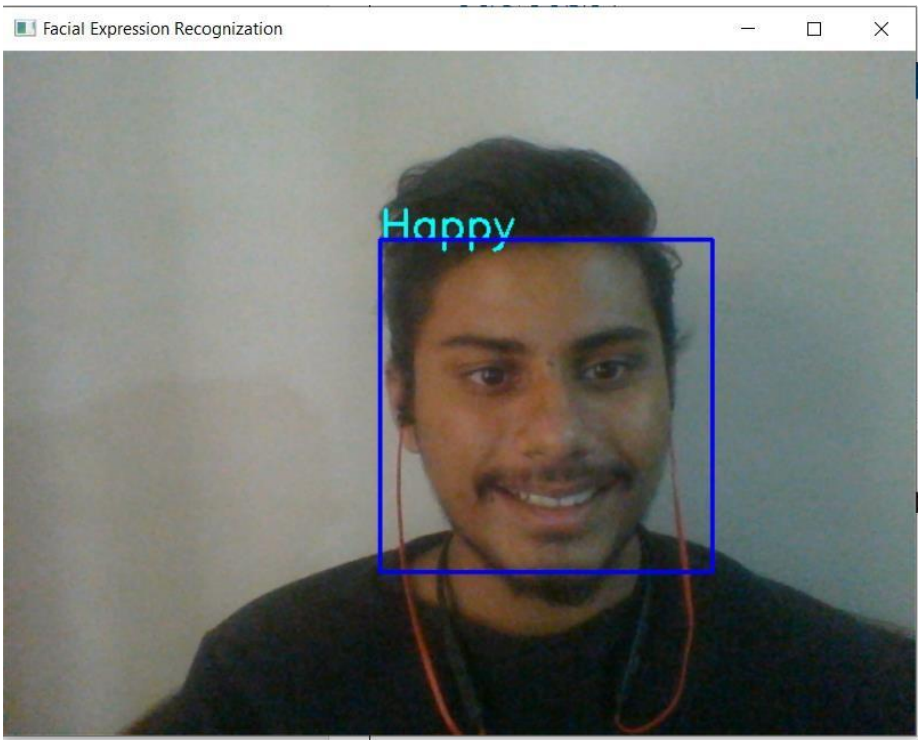
```
gen(VideoCamera())
```

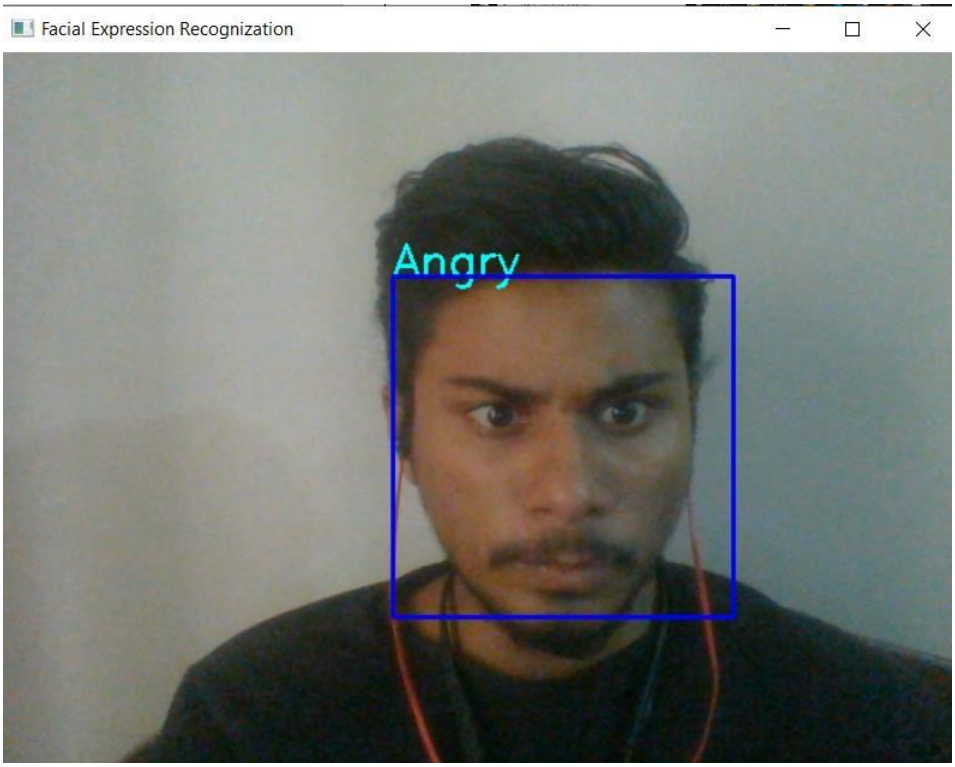
In the above code we have called the gen function and we have passed the VideoCamera class object as the parameter and finally when we run the above line of code the output screen will open showing the live prediction.

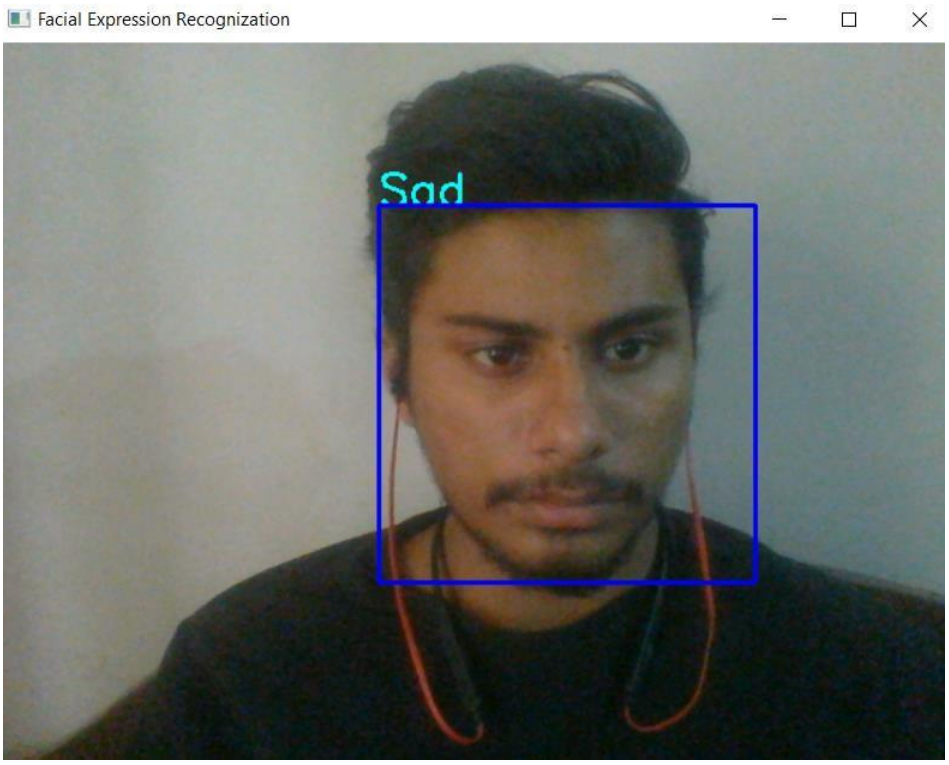
TESTING

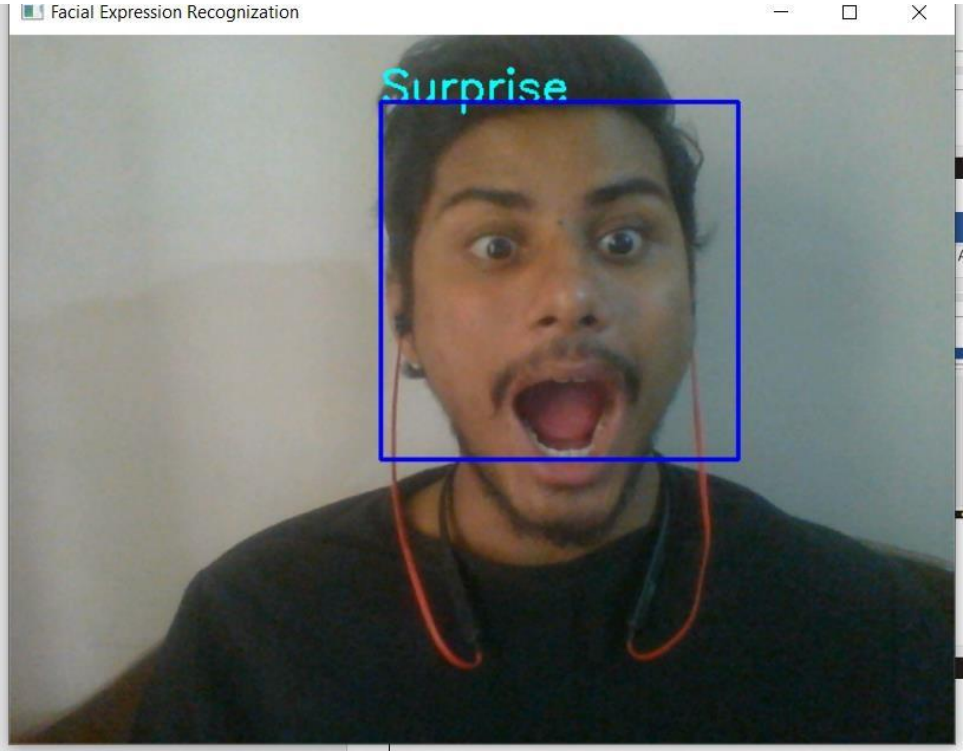
The Following are the facial expressions detected using this model :






Facial Expression Recognization
Angry


Facial Expression Recognization
Sad



After all these predictions and analysis, we can say that the model is working fine with the facial expressions given and the model is accurate enough to predict the facial expression. Based on these expressions the model concludes whether the expression is happy, sad, neutral, angry, surprise, etc.

SCOPE OF IMPROVEMENT

First of all emotion detection is a very important task for many companies to understand how are their consumers reacting to the products launched by them. Also it can be used to know whether their employees are satisfied with the facilities given to them. Also it has many other use cases like checking the mood of a person without getting near to him as we are using camera to detect. Also the same algorithm just needs a little modification and can be used in other fields like face detection, attendance system, mask detection and many more.

REFERENCES

- https://medium.com/analytics-vidhya/facial-expression-detection-using-machine-learning-in-python-c6a188ac765f
- https://colab.research.google.com/
- https://jupyter-notebook.readthedocs.io/en/latest/notebook.html#introduction