



INDUSTRIAL PROJECT REPORT



Submitted in partial fulfilment for the degree of

B-tech in Computer Science Engineering

By

ARUNANGSHU NAG [11900121026]

DIGANTA DAS [11900121014]

SUBARNA GHOSH [11900121036]

ANIRUDDHA GUPTA [11900121021]

ABHIRUP BASU [11900121033]

TRIDIB TALUKDAR [11900121001]

Second - year students of

SILIGURI INSTITUTE OF TECHNOLOGY

AFFILIATED TO

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

Under the supervision of :- Mr. Ripam Kundu



Sikharthy Infotech Pvt. Ltd.

PROJECT ON:- AI DRIVEN SENTIMENT ANALYSIS

By

ARUNANGSHU NAG [11900121026]

DIGANTA DAS [11900121014]

SUBARNA GHOSH [11900121036]

ANIRUDDHA GUPTA [11900121021]

ABHIRUP BASU [11900121033]

TRIDIB TALUKDAR [11900121001]

UNDER THE GUIDANCE OF

Mr. Ripam Kundu

Project Guide



Sikharthy Infotech Pvt. Ltd.

THIS IS SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

B-tech in Computer Science Engineering
SILIGURI INSTITUTE OF TECHNOLOGY

AFFILIATED TO

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

Department of Computer science & Engineering

I hereby forward the documentation prepared under the supervision of **Mr. Ripam Kundu** entitled **Siliguri Institute Of Technology** to be accepted as fulfillment of the requirement for the Degree of Bachelor of Technology in Computer science & Engineering, **Siliguri Institute Of Technology** affiliated to **Maulana Abul Kalam Azad University of Technology (MAKAUT)**.

Mr. Ripam Kundu
(Software Developer)
Project Guide
Sikharthy Infotech Pvt. Ltd.

Shilpi Ghosal
(Director)
Sikharthy Infotech Pvt. Ltd

HEAD OF THE DEPARTMENT

Department Of Computer Science
Engineering, SIT

TPO
Siliguri Institute of Technology

CERTIFICATE OF APPROVAL

The foregoing project is hereby approved as a creditable study for the B.Tech in Computer Science & Engineering presented in a manner of satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorsed or approved any statement made, opinion expressed or conclusion therein but approve this project only for the purpose for which it is submitted.

Final Examination for
Evaluation of the Project

Signatures of Examiners

ABSTRACT

***AI driven Sentiment analysis** or opinion mining is the computational study of people's opinions, sentiments, attitudes, and emotions expressed in written language. It is one of the most active research areas in natural language processing and text mining in recent years. Its popularity is mainly due to two reasons. First, it has a wide range of applications because opinions are central to almost all human activities and are key influencers of our behaviours. Whenever we need to make a decision, we want to hear others' opinions. Second, it presents many challenging research problems, which had never been attempted before the year 2000. Part of the reason for the lack of study before was that there was little opinionated text in digital forms. It is thus no surprise that the inception and the rapid growth of the field coincide with those of the social media on the Web. In fact, the research has also spread outside of computer science to management sciences and social sciences due to its importance to business and society as a whole. In this talk, I will start with the discussion of the mainstream sentiment analysis research and then move on to describe some recent work on modeling comments, discussions, and debates, which represents another kind of analysis of sentiments and opinions.*

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and participation of a large number of individuals in this attempt. Our project report has been structured under the valued suggestion, support, and guidance of **Mr. Ripam Kundu**. Under his guidance, we have accomplished the challenging task in a very short time.

Finally, we express our sincere thankfulness to our family members for inspiring us all throughout and always encouraging us.

Group Members Signatures :

TABLE OF CONTENTS

	PAGE NO.
➤ OVERVIEW OF THE PROJECT	8
➤ PROBLEM STATEMENT, SOLUTION APPROACH, CHALLENGES	9
➤ TEXT EMOTION PREDICTION ANALYSIS	9-12
○ INTRODUCTION	9
○ IMPORTING LIBRARIES	9
○ DATASET	9
○ OPERATIONS OF THE DATASET	10-11
○ DEFINING & TRAINING THE MODEL	11-12
○ PROCESSING THE INPUT TEXT	12
○ GRAPHS	
○ CODE AND OUTPUTS SCREENSHOTS	
➤ FACE EMOTION RECOGNITION	13-19
○ INTRODUCTION	13
○ IMPORTING LIBRARIES	13
○ DATASET	13
○ OPERATIONS OF DATASET	13-15
○ MODEL	15-17
○ PREDICTING THE EMOTION OF AN IMAGE	17-18
○ TEST EMOTION USING WEBCAM	18-19
○ CODE AND OUTPUTS SCREENSHOTS	
➤ FUNCTIONAL REQUIREMENTS	20
➤ CONCLUSION	21
➤ REFERENCE AND BIBLIOGRAPHY	22

OVERVIEW OF THE PROJECT

In this project, our main motive is to create a python program that will detect emotion of a sentence/text and human face. This might be interesting if you want to do things like emotion detection using python, or if you're training machine learning systems to read human emotions. We're going to create programs that takes an image/text as an input and a list of human emotions that the image/text invokes. To do this, we're going to use TensorFlow and other required libraries. Our main motive is to use this project to detect the sentiment analysis to text as well as human face. This project will help to analyse the emotions of a person in the web.

PROBLEM STATEMENT

- **Lack of proper datasets**
- **Short time duration for completion**
- **Requirement of high configuration machines**
- **Proper output from a small datasets**

SOLUTION APPROACH

- **Searched and created proper datasets for proper functioning of the program with the machines**
- **Good team cooperation.**

CHALLENGES

- **Need of high configuration machines for more accurate detection.**

TEXT EMOTION PREDICTION ANALYSIS

INTRODUCTION:

For text emotion prediction analysis we are using python and its other required libraries which will complete the prediction. Our objective is to predict the correct emotion of a desired input text.

IMPORTING LIBRARIES:

HERE WE NEED DIFFERENT TYPES OF LIBRARIES FOR THE ANALYSIS. THE LIBRARIES THE LISTED BELOW:

- **PANDAS:** This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- **NUMPY:** NumPy arrays are very fast and can perform large computations in a very short time.
- **TENSORFLOW:** TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.
- **SCIKIT-LEARN:** It is a Python library is associated with NumPy and SciPy. It is considered as one of the best libraries for working with complex data.
- **KERAS:** Keras is considered as one of the coolest machine learning libraries in Python. It provides an easier mechanism to express neural networks. Keras also provides some of the best utilities for compiling models, processing data-sets, visualization of graphs, and much more.
- **MATPLOTLIB:** Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- **SEABORN:** seaborn is a python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

CODE TO IMPORT LIBRARIES:

```
import pandas as pd
import numpy as np
import keras
import tensorflow
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow import keras
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
```

DATASET:

For this analysis we are using "text.txt" dataset. This dataset is inserted after importing the libraries.

CODE:

IMPORTING THE REQUIRED DATASET

```
1 data = pd.read_csv("train.txt", sep=';')
2 data.columns = ["Text", "Emotions"]
3 print(data.head())
```

OUTPUT AFTER IMPORTING THE DATASET:

	Text	Emotions
0	i can go from feeling so hopeless to so damned...	sadness
1	im grabbing a minute to post i feel greedy wrong	anger
2	i am ever feeling nostalgic about the fireplac...	love
3	i am feeling grouchy	anger
4	ive been feeling a little burdened lately wasn...	sadness

OPERATIONS OF THE DATASET:

COUNTING THE DATAS PER EMOTION LABEL IN THE DATSET:

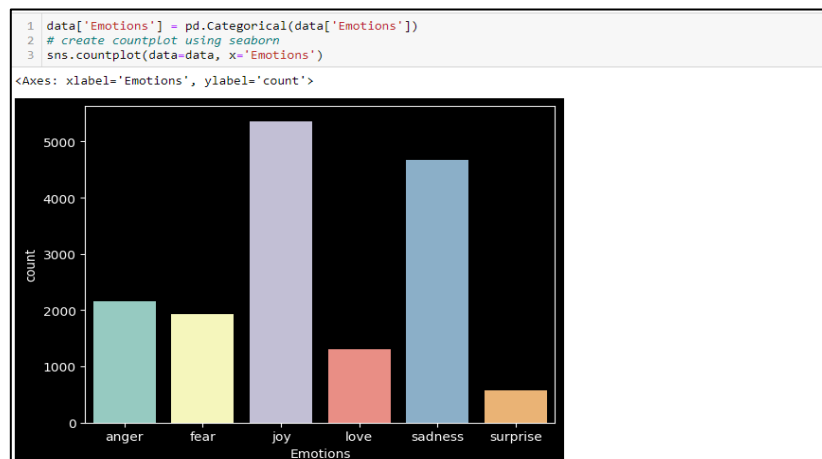
CODE AND OUTPUT:

COUNTING THE DATAS PER EMOTION LABELS	
In [3]:	1 data['Emotions'].value_counts()
Out[3]:	joy 5362 sadness 4665 anger 2159 fear 1937 love 1304 surprise 572 Name: Emotions, dtype: int64

PLOTTING THE GRAPH OF DATA vs. EMOTIONS:

Here we are plotting the bar graph using the seaborn library which represents the no. of text data per emotion labels.

CODE AND OUTPUT:



LISTING THE COLUMNS OF THE DATASET:

Here we are converting the columns of the dataset into list.

CODE:

```
1 texts = data["Text"].tolist()
2 labels = data["Emotions"].tolist()
```

TOKENIZING THE TEXT DATA:

In Python tokenization basically refers to splitting up a larger body of text into smaller lines, words or even creating words for a non-English language.

CODE:

```
1 tokenizer = Tokenizer()
2 tokenizer.fit_on_texts(texts)

1 sequences = tokenizer.texts_to_sequences(texts)
2 max_length = max([len(seq) for seq in sequences])
3 padded_sequences = pad_sequences(sequences, maxlen=max_length)
```

LABEL ENCODING OF DATASET:

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form.

CODE:

```
1 one_hot_labels = keras.utils.to_categorical(labels)
```

SPLITTING THE DATA :

Here we are splitting the data into the training and testing sets. Train test split is a model validation process that allows you to simulate how your model would perform with new data. Split the data set into two pieces — a training set and a testing set. This consists of random sampling without replacement about 75 percent of the rows (you can vary this) and putting them into your training set. The remaining 25 percent is put into your test set.

CODE:

```
xtrain, xtest, ytrain, ytest = train_test_split(padded_sequences,
                                                one_hot_labels,
                                                test_size=0.2)
```

DEFINING AND TRAINING THE MODEL

A model is a python class that inherits from the model class. The model class defines a new kind of datastore entity and the properties the kind is expected to take. Here we are training the model with epoch=100. And subsequently our model is saved as "hlo.h5".

CODE AND OUTPUT:

```
1 model = Sequential()
2 model.add(Embedding(input_dim=len(tokenizer.word_index) + 1,
3                     output_dim=128, input_length=max_length))
4 model.add(Flatten())
5 model.add(Dense(units=128, activation="relu"))
6 model.add(Dense(units=len(one_hot_labels[0]), activation="softmax"))
7
8 model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
9 model.fit(xtrain, ytrain, epochs=100, batch_size=32, validation_data=(xtest, ytest))

Epoch 95/100
400/400 [-----] - 10s 25ms/step - loss: 0.0021 - accuracy: 0.9980 - val_loss: 3.2053 - val_accuracy:
0.7584
Epoch 96/100
400/400 [-----] - 10s 25ms/step - loss: 0.0021 - accuracy: 0.9984 - val_loss: 3.2072 - val_accuracy:
0.7578
Epoch 97/100
400/400 [-----] - 10s 24ms/step - loss: 0.0021 - accuracy: 0.9981 - val_loss: 3.2129 - val_accuracy:
0.7569
Epoch 98/100
400/400 [-----] - 10s 25ms/step - loss: 0.0021 - accuracy: 0.9980 - val_loss: 3.2090 - val_accuracy:
0.7572
Epoch 99/100
400/400 [-----] - 10s 25ms/step - loss: 0.0021 - accuracy: 0.9985 - val_loss: 3.2152 - val_accuracy:
0.7569
Epoch 100/100
400/400 [-----] - 10s 25ms/step - loss: 0.0130 - accuracy: 0.9963 - val_loss: 3.5587 - val_accuracy:
0.7400
: <keras.callbacks.History at 0x1f9a526e500>
```

PLOTTING THE GRAPH OF LOSS AND ACCURACY:

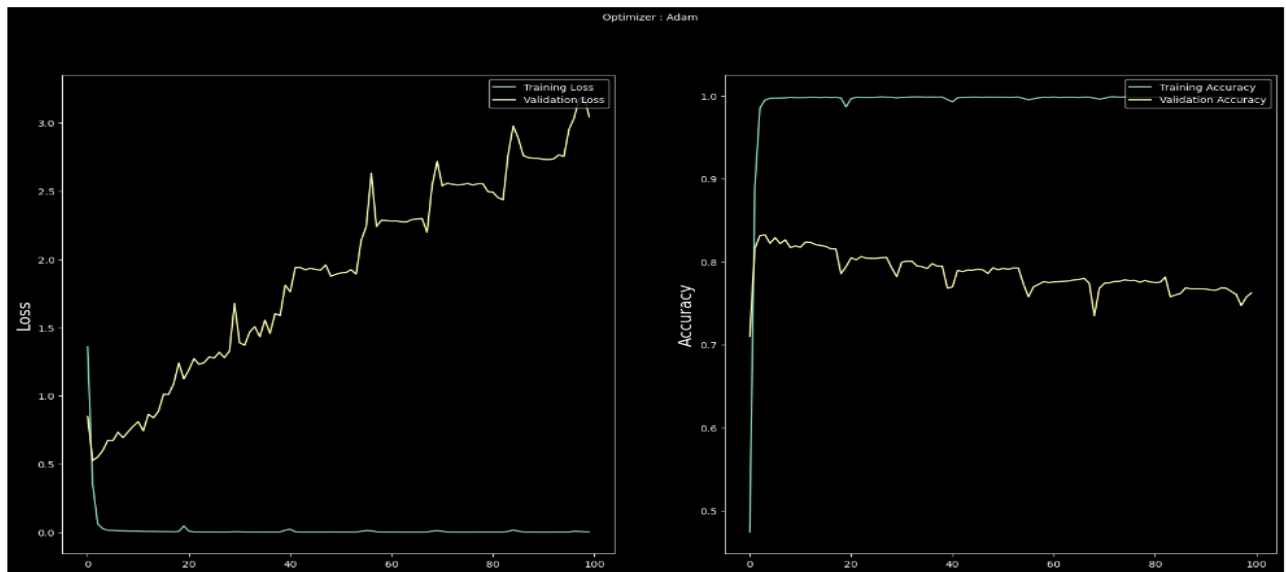
Here we are plotting the graph of loss and accuracy after training the model. The graphs plots the training and validation loss and accuracy of both the fields.

CODE:

```
plt.style.use('dark_background')
plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(model.history.history['loss'], label='Training Loss')
plt.plot(model.history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(model.history.history['accuracy'], label='Training Accuracy')
plt.plot(model.history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='upper right')
plt.show()
```

OUTPUT (GRAPH):



SAVING AND LOADING THE MODEL:

After training the model we are saving the model as “hlo.h5” and subsequently then loading the model using tensorflow.keras.models library.

CODE:

```
model.save("hlo1.h5") #saving the model
: model=tensorflow.keras.models.load_model('hlo1.h5')
```

PROCESSING THE INPUT TEXT

In this part we enter a desired text and then by processing the input text by the train model and hence we receive the subsequent output.

CODE:

```
input_text = input("")
# Preprocess the input text
input_sequence = tokenizer.texts_to_sequences([input_text])
padded_input_sequence = pad_sequences(input_sequence, maxlen=max_length)
prediction = model.predict(padded_input_sequence)
predicted_label = label_encoder.inverse_transform([np.argmax(prediction[0])])
print(predicted_label)
```

OUTPUT:

Input text: “I lost my pen”

Original output should be sad and predicted output is also sadness.

```
I LOST MY PEN
1/1 [=====] - 0s 27ms/step
['sadness']
```

FACE EMOTION RECOGNITION

INTRODUCTION:

For face emotion recognition we are using python and its other required libraries which will complete the prediction. Our objective is to predict the correct emotion of a real time face using webcam.

IMPORTING LIBRARIES:

HERE WE NEED DIFFERENT TYPES OF LIBRARIES FOR THE ANALYSIS. THE LIBRARIES THE LISTED BELOW:

- **PANDAS**: This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- **NUMPY**: NumPy arrays are very fast and can perform large computations in a very short time.
- **TENSORFLOW**: TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.
- **SCIKIT-LEARN**: It is a Python library is associated with NumPy and SciPy. It is considered as one of the best libraries for working with complex data.
- **KERAS**: Keras is considered as one of the coolest machine learning libraries in Python. It provides an easier mechanism to express neural networks. Keras also provides some of the best utilities for compiling models, processing data-sets, visualization of graphs, and much more.
- **MATPLOTLIB**: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- **OPEN-CV**: OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability.
- **RANDOM**: The Python Random module is a built-in module for generating random integers in Python.

CODE TO IMPORT LIBRARIES:

```
1 import tensorflow as tf
2 import cv2
3 import os
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import pandas as pd
7 import tensorflow as tf
8 from tensorflow import keras
9 from tensorflow.keras import layers
```

DATASET:

For this analysis we are using the directory "train/" dataset. This dataset is inserted after importing the libraries. The folders of the dataset are inserted as a list under the classes. The dataset contains approx. 3000 images for training.

CODE:

```
1 Datadirectory= "train/"
1 Classes= ["0","1","2","3","4","5","6"] #CLASSES REFERS TO THE FOLDERS OF THE TRAIN DIRECTORY
```

CLASSES 0=ANGRY, 1=DISGUST, 2=FEAR,3= HAPPY 4= NEUTRAL, 5= SAD, 6 = SURPRISE

OPERATIONS OF THE DATASET:

Here we will resize, reshape the images , create category, creating the training data, shuffling the data randomly, Labelling the data, normalizing pixels of the images.

WORKING WITH A PRE LOADED IMAGE FROM DATASET:

Loading a training image from the directory to perform and check the operations of the dataset.

CODE AND OUTPUT:

```
In [5]: 1 img_array = cv2.imread("train/0/Training_3908.jpg") #LOADING A TRAINING IMAGE

In [6]: 1 img_array.shape #PRINTING THE SHAPE OF THE LOADED IMAGE
Out[6]: (48, 48, 3)

In [7]: 1 plt.imshow(img_array) #PRINTING THE IMAGE AS ARRAY
Out[7]: <matplotlib.image.AxesImage at 0x22bcae88940>
```




RESIZING THE IMAGE DATASET:

The **ImageNet** dataset contains images of fixed **size** of **224*224** and **have** RGB channels but as **fer2013** has images of size **48*48** so we'll have to resize the images. To resize an image, OpenCV provides `cv2.resize()` function. `cv2.cvtColor()` method is used to convert an image from one color space to another. The reason behind executing the above code is that we're using transfer learning so for transfer learning if we want to use any deep learning classifier then these dimensions must be same.

CODE AND OUTPUT:

```
1 img_size = 224
2 new_array = cv2.resize(img_array,(img_size,img_size))
3 plt.imshow( cv2.cvtColor(new_array,cv2.COLOR_BGR2RGB))
4 plt.show()
```



```
1 new_array.shape #SHAPE AFTER RESIZING THE IMAGE
(224, 224, 3)
```

CONVERTING THE IMAGES INTO ARRAY:

Here we are converting the images into array and storing in "training_data" and counting the total no. of data. And calling the function.

CODE AND OUTPUT:

```
1 training_Data = []
2 def create_training_Data():
3     for category in Classes:
4         path = os.path.join(DataDirectory, category)
5         class_num = Classes.index(category)
6         for img in os.listdir(path):
7             try:
8                 img_array = cv2.imread(os.path.join(path, img))
9                 new_array = cv2.resize(img_array, (img_size, img_size))
10                training_Data.append([new_array, class_num])
11            except Exception as e:
12                pass
```

```
1 create_training_Data()
```

```
1 print(len(training_Data))
```

```
2948
```

SHUFFLING THE DATA:

To make our deep learning architecture dynamic and robust, let's shuffle the sequence.

CODE:

```
1 import random
2 random.shuffle(training_Data)
```

SEPARATING FEATURES & LABELS:

Let's separate the features and labels. We'll use deep learning architecture MobileNetV2 which takes 4 dimensions, so we'll reshape the features list.

CODE:

```
In [19]: 1 X=[]
2 y = []
3 for features, label in training_Data:
4     X.append(features)
5     y.append(label)
6 X = np.array(X).reshape(-1, img_size, img_size, 3)
7 # 3 is the channel for RGB
```

```
In [20]: 1 X.shape #PRINTING THE SHAPE OF THE FEATURES
```

```
Out[20]: (2948, 224, 224, 3)
```

```
In [22]: 1 y[900]
```

```
Out[22]: 2
```

```
In [23]: 1 Y=np.array(y)
```

```
In [24]: 1 Y.shape
```

```
Out[24]: (2948,)
```

NORMALIZING THE DATA:

Among the best practices for **training** a Neural Network is to **normalize** your data to obtain a mean close to 0. **Normalizing** the data generally speeds up learning and leads to faster convergence. Let's normalize the data before training.

CODE:

```
In [21]: 1 X = X/255.0 ;
```

MODEL

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. Now we'll use MobileNetV2.

CODE OF DEFINING THE MODEL:

```
: 1 model = tf.keras.applications.MobileNetV2() #USING MOBILENETV2 APPLICATION FOR MODELLING THE DATA
: 1 model.summary()
Model: "mobilenetv2_1.00_224"
-----
Layer (type)                 Output Shape          Param #   Connected to
-----
input_1 (InputLayer)         [(None, 224, 224, 3  0
)]
Conv1 (Conv2D)                (None, 112, 112, 32  864
)
bn_Conv1 (BatchNormalization) (None, 112, 112, 32  128
)
Conv1_relu (ReLU)             (None, 112, 112, 32  0
)
expanded_conv_depthwise (Depth (None, 112, 112, 32  288
wiseConv2D)
)
```

Let's change the base input and As we want seven classes, so let's cut down output.

```
1 base_input= model.layers[0].input
```

```
1 base_output= model.layers[-2].output
```

WORKING OF DENSE LAYERS OF THE MODEL:

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The activation function is a mathematical “gate” in between the input feeding the current neuron and its output going to the next layer. It can be as simple as a step function that turns the neuron output on and off, depending on a rule or threshold. Here we're using relu as activation function.

CODE:

```
final_output = layers.Dense(128)(base_output)
final_output = layers.Activation('relu')(final_output)
final_output = layers.Dense(64)(final_output)
final_output = layers.Activation('relu')(final_output)
final_output = layers.Dense(7,activation='softmax')(final_output)
```

CREATING THE NEW MODEL AND COMPILING IT:

Compile defines the loss function, the optimizer, and the metrics. That's all. It has nothing to do with the weights and you can compile a model as many times as you want without causing any problem to pre-trained weights. You need a compiled model to train (because training uses the loss function and the optimizer). Here we are training the model with 50 epochs.

CODE AND OUTPUT:

```
new_model = keras.Model(inputs = base_input, outputs=final_output)
```

```
In [34]: 1 new_model.compile(loss="sparse_categorical_crossentropy",optimizer = "adam", metrics = ["accuracy"])
```

```
In [35]: 1 history=new_model.fit(X,Y, epochs=50)
```

```
93/93 [=====] - 166s 2s/step - loss: 0.1762 - accuracy: 0.9444
Epoch 42/50
93/93 [=====] - 168s 2s/step - loss: 0.0725 - accuracy: 0.9769
Epoch 43/50
93/93 [=====] - 166s 2s/step - loss: 0.0641 - accuracy: 0.9786
Epoch 44/50
93/93 [=====] - 167s 2s/step - loss: 0.0722 - accuracy: 0.9752
Epoch 45/50
93/93 [=====] - 167s 2s/step - loss: 0.0763 - accuracy: 0.9763
Epoch 46/50
93/93 [=====] - 168s 2s/step - loss: 0.1689 - accuracy: 0.9467
Epoch 47/50
93/93 [=====] - 164s 2s/step - loss: 0.1051 - accuracy: 0.9671
Epoch 48/50
93/93 [=====] - 164s 2s/step - loss: 0.1229 - accuracy: 0.9640
Epoch 49/50
93/93 [=====] - 162s 2s/step - loss: 0.1829 - accuracy: 0.9427
Epoch 50/50
93/93 [=====] - 162s 2s/step - loss: 0.0975 - accuracy: 0.9746
```

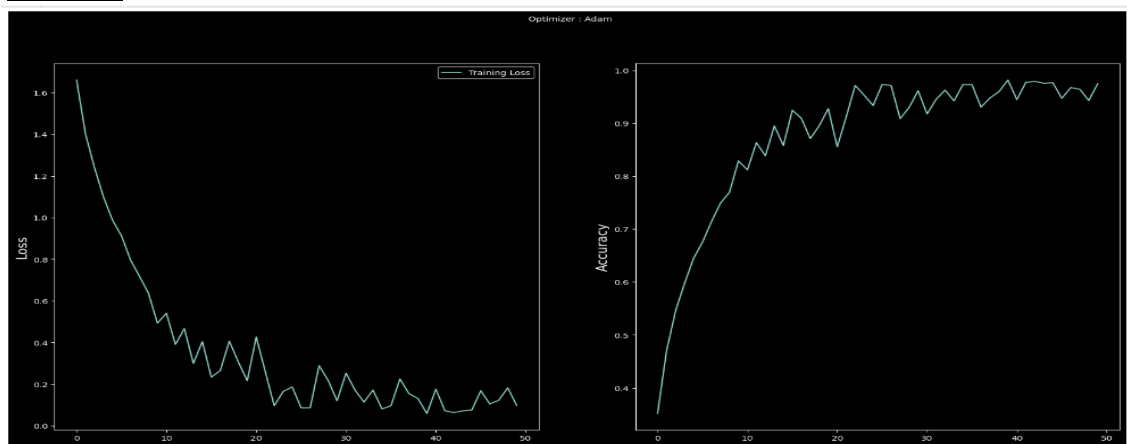
PLOTTING THE GRAPH:

After training the model we will plot the loss and accuracy graph.

CODE:

```
1 plt.style.use('dark_background')
2
3 plt.figure(figsize=(20,10))
4 plt.subplot(1, 2, 1)
5 plt.title('Optimizer : Adam', fontsize=10)
6 plt.ylabel('Loss', fontsize=16)
7 plt.plot(new_model.history.history['loss'], label='Training Loss')
8
9 plt.legend(loc='upper right')
10 plt.subplot(1, 2, 2)
11 plt.ylabel('Accuracy', fontsize=16)
12 plt.plot(new_model.history.history['accuracy'], label='Training Accuracy')
13
14 plt.show()
15
```

GRAPH:



SAVING AND LOADING THE MODEL:

After training the model we are saving the model as “mymodel.h5” and subsequently then loading the model using tensorflow.keras.models library.

```
: 1 new_model.save("mymodel.h5")
: 1 new_model = tf.keras.models.load_model('mymodel.h5') #LOADING THE TRAINED MODEL
```

PREDICTING THE EMOTION OF AN IMAGE:

Here we are inserting a image and predicting the image emotion through our trained model.

INSERTING AN IMAGE:

We are inserting a image and changing it to
Grayscale

CODE AND OUTPUT:

```
1 frame = cv2.imread("neutral.jpg") #IMPORTING AN IMAGE
2 frame.shape #PRINING THE SHAPE OF THE IMAGE
(116, 174, 3)
1 plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
<matplotlib.image.AxesImage at 0x22d45efa320>
```



INSERTING THE HAARCASCADE FILE:

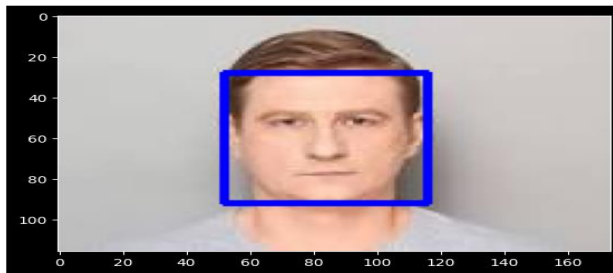
Here we are inserting the "haarcascade.defaultface.xml" file which helps to detect the face from an image

```
1 faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

CODE FOR PREDICTION OF EMOTION AND FACE FROM THE IMAGE:

```
1 faces = faceCascade.detectMultiScale(gray,1.1,4)
2 for x,y,w,h in faces:
3     roi_gray = gray[y:y+h, x:x+w]
4     roi_color = frame[y:y+h, x:x+w]
5     cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
6     faces = faceCascade.detectMultiScale(roi_gray)
7     if len(faces) == 0:
8         print("face not detected")
9     else:
10         for (ex,ey,ew,eh) in faces:
11             face_roi=roi_color[ey:ey+eh,ex:ex+ew]
```

```
1 plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)) #PLOTING THE IMAGE
<matplotlib.image.AxesImage at 0x22d4712e8c0>
```



```
1 # RESIZING AND NORMALIZING THE IMAGE
2 final_image= cv2.resize(frame,(224,224))
3 final_image= np.expand_dims(final_image,axis=0)
4 final_image=final_image/255.0
```

PREDICTING THE EMOTION AND RESULT:

```
1 Predictions = new_model.predict(final_image) #PREDICTING THE FINAL IMAGE
```

```
1/1 [=====] - 0s 32ms/step
```

```
1 Predictions[0]
```

```
array([1.0216828e-05, 6.9333182e-05, 2.8691704e-03, 1.7691565e-03,
        9.9040633e-01, 1.3542562e-05, 4.8621870e-03], dtype=float32)
```

```
1 np.argmax(Predictions) #PREDICTING THE EMOTION
```

```
4
```

As we can see the predicted class is 4 therefore, we know 4 refers to neutral images so it predicted a correct emotion.

TEST EMOTION USING WEBCAM:

For accessing the webcam we will use the opencv, tensorflow and keras libraries and detect face using the haarcascade_defaultface.xml file.

CODE:

```
import cv2
import numpy as np
import tensorflow as tf
from keras.applications.mobilenet_v2 import preprocess_input
from keras.preprocessing.image import img_to_array

# Load pre-trained model
model = tf.keras.models.load_model('mymodel.h5')

# Define emotion labels
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']

# Initialize OpenCV face detector
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
#face_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')
#face_cascade = cv2.CascadeClassifier('haarcascade_frontalcatface.xml')

# Initialize camera
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error opening video stream or file")

while True:
    # Read image from camera
    ret, img = cap.read()
    if ret:
        # Convert image to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Detect faces in the image
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)

        # Process each detected face
        for (x,y,w,h) in faces:
            # Extract face region of interest
            roi_gray = gray[y:y+h, x:x+w]
            roi_color = img[y:y+h, x:x+w]

            # Resize face image to match input shape of model
            roi_gray = cv2.resize(roi_gray, (224, 224), interpolation=cv2.INTER_AREA)
            roi_gray = roi_gray.reshape(1, 224, 224, 1)
            # Convert grayscale image to RGB
            cv2.cvtColor(roi_color, cv2.COLOR_BGR2RGB)

            # Resize image to 224x224 and preprocess for model
            roi_color = cv2.resize(roi_color, (224, 224))
            roi_color = np.array(roi_color)
            roi_color = np.expand_dims(roi_color, axis=0)
            roi_color = preprocess_input(roi_color)

            # Predict emotion using pre-trained model
            prediction = model.predict(roi_color)
            emotion = emotion_labels[np.argmax(prediction)]

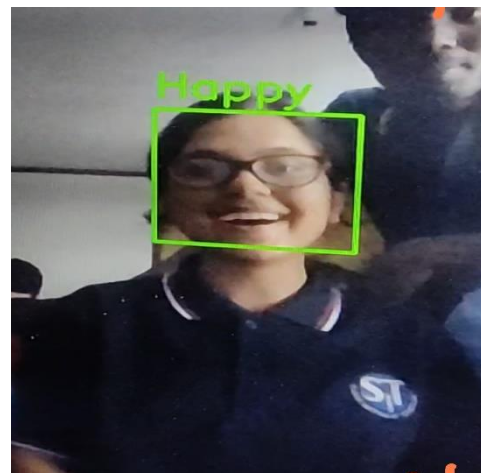
            # Normalize pixel values
            roi_gray = roi_gray / 255.0
            # Predict emotion using pre-trained model
            #prediction = model.predict(roi_gray)
            #emotion = emotion_labels[np.argmax(prediction)]
            prediction = model.predict(roi_color)
            emotion = emotion_labels[np.argmax(prediction)]
            # Draw bounding box and label on face
            cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
            cv2.putText(img, emotion, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

        # Display output image
        cv2.imshow('Face Emotion Recognition', img)

        # Break loop when 'q' key is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        print("Error reading video stream")
        break
# Release resources
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

Snippet of testing the face and detecting
“happy”



FUNCTIONAL REQUIREMENTS

SOFTWARE:

- WINDOWS 11

WEB BROWSER

- GOOGLE CHROME
- MOZILLA FIREFOX

CODING LANGUAGE

- PYTHON(3.10.8)

CONCLUSION

AS THE PROJECT HAS TWO PARTS: TEXT DETECTION AND FACE EMOTION DETECTION WE FACED MANY CHALLENGES TO PREDICT THE CORRECT RESULT. AS FOR DETECTION OF AN IMAGE/FACE WE HAD A VERY BIG DATASET , WHICH REQUIRED HIGH CONFIGURATION MACHINES SO WE HAD TO REDUCE OUR DATASET. SUBSEQUENTLY THE ACCURACY WAS NOT ACHIEVED AND HENCE HAD TO FACE A LOT OF CHALLENGES FOR IT. BUT AFTER TRAINING THE MODELS PROPERLY OUR TEXT EMOTION DETECTION IS WORKING APPROX 80% CORRECT EMOTION DETECTION AND AS WE HAVE A VERY SMALL DATASET OF IMAGES SO WE HAVE ACHIEVED 60% APPROX ACCURACY FOR THE MODEL TO DETECT THE EMOTION. THIS PROJECT WILL HELP US TO DETECT THE SENTIMENT OF SOMEONE IN THE WEB (SOCIAL MEDIA) THROUGH TEXT AND IN FUTURE THE FACE DETECTION WILL HELP TO REVEAL INFORMATION ON ONE'S EMOTIONAL STATE.

REFERENCES AND BIBLIOGRAPHY

- www.geesksforgeesks.com
- medium.com/analytics-vidhya/realtime-face-emotion-recognition-using-transfer-learning-in-tensorflow-3add4f4f3ff3
- www.github.com
- www.kaggle.com/datasets/msambare/fer2013
- www.wikipedia.org