NEURAL NETWORK TECHNIQUE FOR DETECTING FACIAL EXPRESSIONS OF EMOTIONS

A PROJECT REPORT

Submitted by

ENDLURU VIGNESH

THATIPIGARI BHAGEERATH

AKSHAY KUMAR KUSHWAHA

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

AITCSE-AIML

Chandigarh University

NOVEMBER 2022

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report **"NEURAL NETWORK TECHNIQUE FOR DETECTING FACIAL EXPRESSIONS OF EMOTIONS"** is the bonafide work of "**ENDLURU VIGNESH, THATIPIGARI BHAGEERATH AND AKSHAY KUMAR KUSHWAHA" who** carried out the project work under my/our supervision.

SIGNATURE **SIGNATURE**

HEAD OF THE DEPARTMENT **SUPERVISOR**

Submitted for the project viva-voce examination held on_

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

List of Figures

ABSTRACT

Emotion Recognition is a task to process a human facial expression and classify it into certain emotion categories. Such task typically requires the feature extractor to detect the feature, and the trained classifier produces the label based on the feature. The problem is that the extraction of feature may be distorted by variance of location of object and lighting condition in the image. In this project, we address the solution of the problem by using a deep learning algorithm called Conventional Neural Network (CNN) to address the issues above. By using this algorithm, the feature of image can be extracted without user-defined feature-engineering, and classifier model is integrated with feature extractor to produce the result when input is given. In this way, such method produces a feature-location-invariant image classifier that achieves higher accuracy than traditional linear classifier when the variance such as lighting noise and background environment appears in the input image. Facial emotion recognition (FER) is an important topic in the fields of computer vision and artificial intelligence owing to its significant academic and commercial potential. Although FER can be conducted using multiple sensors, this review focuses on studies that exclusively use facial images, because visual expressions are one of the main information channels in interpersonal communication. The evaluation of the model shows that the accuracy of our lab condition testing data set is 82.04%.

GRAPHICAL ABSTRACT

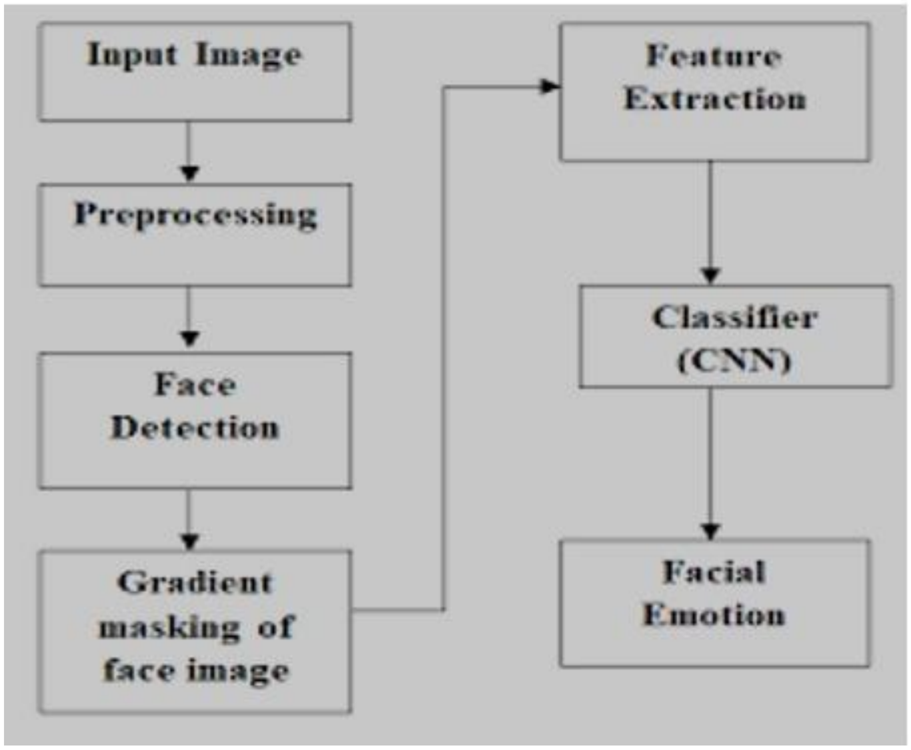


Figure i

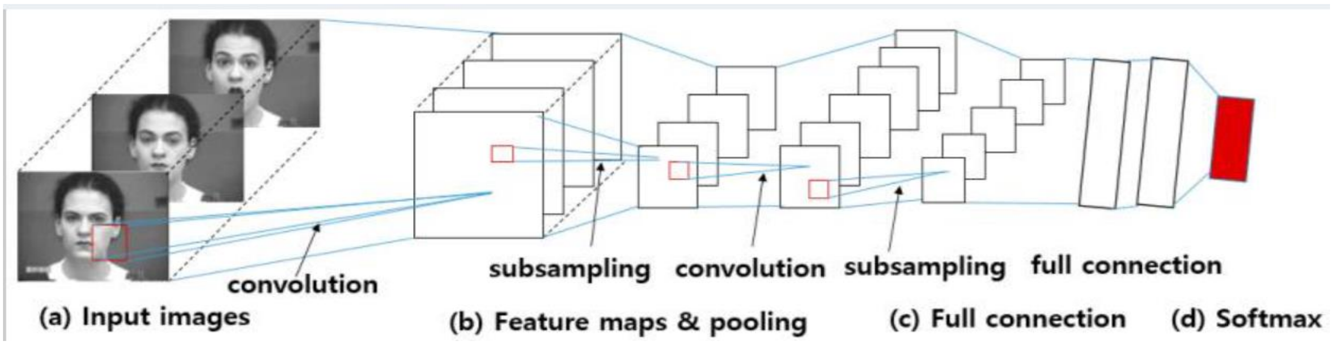


Figure ii

CHAPTER 1.
INTRODUCTION

FACIAL expressions play a key role in understanding and detecting emotion. Even the term "interface" suggests how important face plays in communication between two entities. Studies have shown that reading of facial expressions can significantly alter the interpretation of what is spoken as well as control the flow of a conversation. The ability for humans to interpret emotions is very important to effective communication; accounting for up to 93% of communication used in a normal conversation depends on the emotion of an entity. For ideal human-computer interfaces (HCI) would desire that machines can read human emotion. For that this research is all about how computers can detect emotion properly from its various sensors. This experiment has been used as a facial image as a medium to read human emotion. The research on human emotion can be traced back to Darwin's pioneer working and since then has attracted a lot of researchers to this area. Seven basic emotions are universal to human beings. Namely neutral, angry, disgust, fear, happy, sad, and surprise, and these basic emotions can be recognized from a human's facial expression. This research proposes an effective way to detect neutral, happy, sad, and surprise these four emotions from frontal facial emotion.

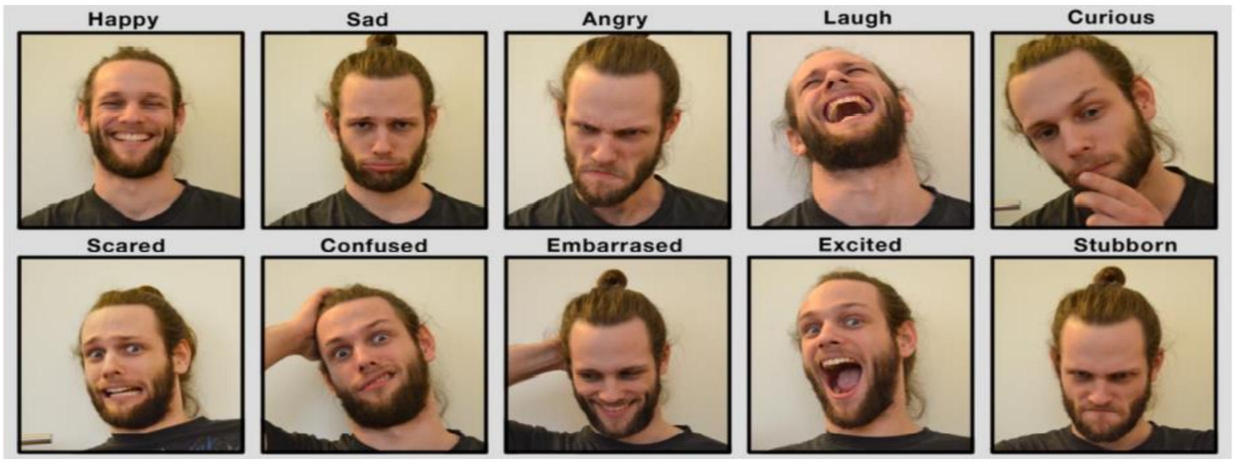


Figure 1

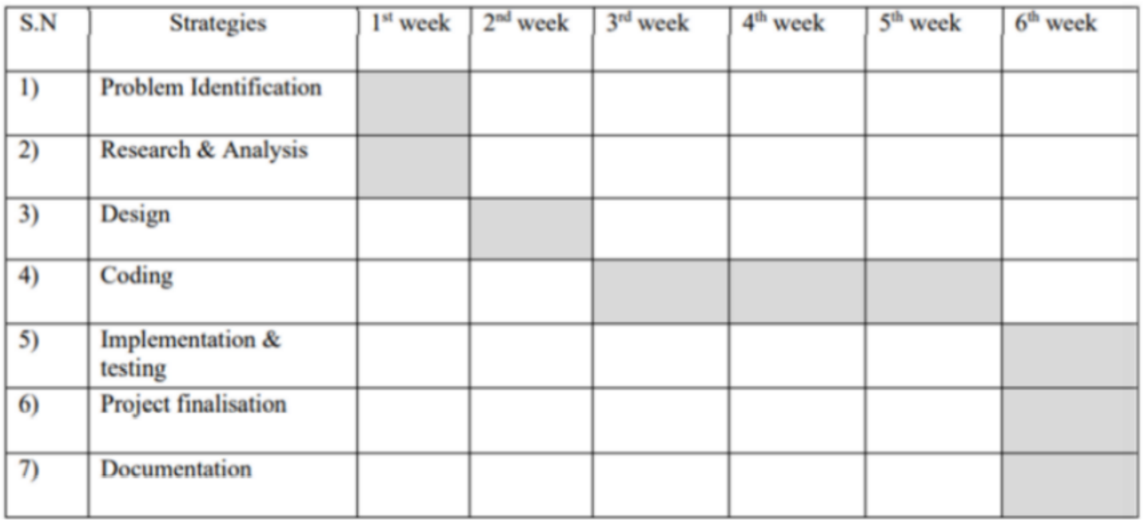
1.1. Problem Definition

Human face is vast with its emotions although its facial structure is simple. Recognizing a human emotion with naked eye is comparatively easy with computer vision. Some algorithms and techniques are used to overcome these problems. There are some set of emotions like (sad, neutral), (disgust, confused) which are similar. These emotions are hard to differentiate even to human brain, but machine learning and neural networks are present to replicate or even ahead of human brain with the help of huge data. However, computer vision also takes raw data, it requires deep refining of data and optimizing it.

1.2. Project Overview

With the emotion recognition system, AI can detect the emotions of a person through their facial expressions. Detected emotions can fall into any of the six main data of emotions: happiness, sadness, fear, surprise, disgust, and anger. For example, a smile on a person can be easily identified by the AI as happiness. Human emotion recognition plays an important role in the interpersonal relationship. The automatic recognition of emotions has been an active research topic from early eras. Therefore, there are several advances made in this field. Emotions are reflected from speech, hand, and gestures of the body and through facial expressions. Hence extracting and understanding of emotion has a high importance of the interaction between human and machine communication. This paper describes the advances made in this field and the various approaches used for recognition of emotions. The main objective of the paper is to propose real time implementation of emotion recognition system.

1.3. Timeline



S.N	Strategies	1st week	2nd week	3rd week	4th week	5th week	6th week
1)	Problem Identification	▨					
2)	Research & Analysis	▨					
3)	Design		▨				
4)	Coding			▨	▨	▨	
5)	Implementation & testing						▨
6)	Project finalisation						▨
7)	Documentation						▨

1.4. Hardware and Software requirements

1) Computer with i3 above or equivalent core chip

2) RAM 6 or above

3) PYTHON

4) NUMPY

5) PANDAS

6) FER2013 DATASET

CHAPTER 2.
LITERATURE REVIEW/BACKGROUND STUDY

In a research field of emotion detection, there is a contribution of several domains like machine learning, natural language, neuroscience, etc. In previous works, they individually rummaged facial expressions, voice features, and textual data as universal indicators of emotions. Emotion can be classified into several static classifications like happiness, sadness, disgust, anger, fear, and surprise. In later works are improved by combining the image, voice, and textual data. The fusion of this data gives the maximum accurate result. This type of fusion can be done in three ways early, late, or hybrid. Other ethos features the elements of emotion and the collaborations between emotional processes and other intellectual procedures.

2.1. Existing System

The existing conventional method requires a handcrafted feature extractor of facial Action Units (AUs) to extract feature from designated Facial Landmark regions, and these extracted AUs codes are processed through traditional machine learning algorithm such as Nearest Neighbors and SVM, which is a typical type of linear classifier. The problem with conventional method is that the lighting 9 variations and different position of object may corrupt the feature vector so that the accuracy is greatly reduced. Furthermore, it is typically difficult to conduct feature-engineering to fit the demand of facial recognition.

2.2. Proposed System

We have been having various optimizers for face detection, analysis of the captured image and its meaningful analysis by facial expressions, creating data sets for test and training and then the designing and the implementation of perfectly fitted classifiers to learn underlying classifiers to learn the vectors of the facial descriptors. We propose a model design which can recognize up to six models which are considered universal among all walks of cultures. Mainly being fear, happiness, sadness, surprise, disgust and lastly surprise. Our system would be to understand a face and its characteristics and then make a weighted assumption of the identity of the person. Some easy differences for complex emotions like sad and neutral are being handled. We created an mobile application by using this trained model.

CHAPTER 3.

DESIGN FLOW/PROCESS

The main modules used in this project are NumPy, Pandas, scikitplot, seaborn, sklearn, tensorflow, Keras layers and some CNN functions.

What is Neural Network?

Neural networks are designed by observing the functionality of neurons in a human brain. In a neural network, there are three layers: Input Layer, Hidden Layers, and Output layer. The input layer consists of the inputs or the independent X variable known as the predictors. These inputs are collected from external sources such as text data, images, audio, or video files.

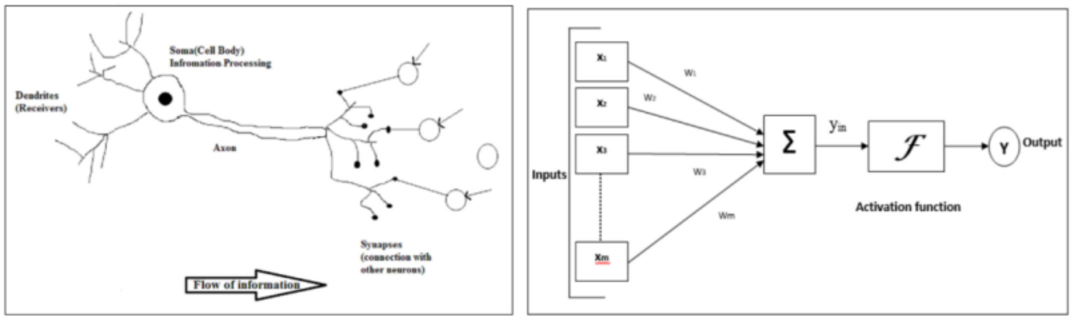


Figure 2

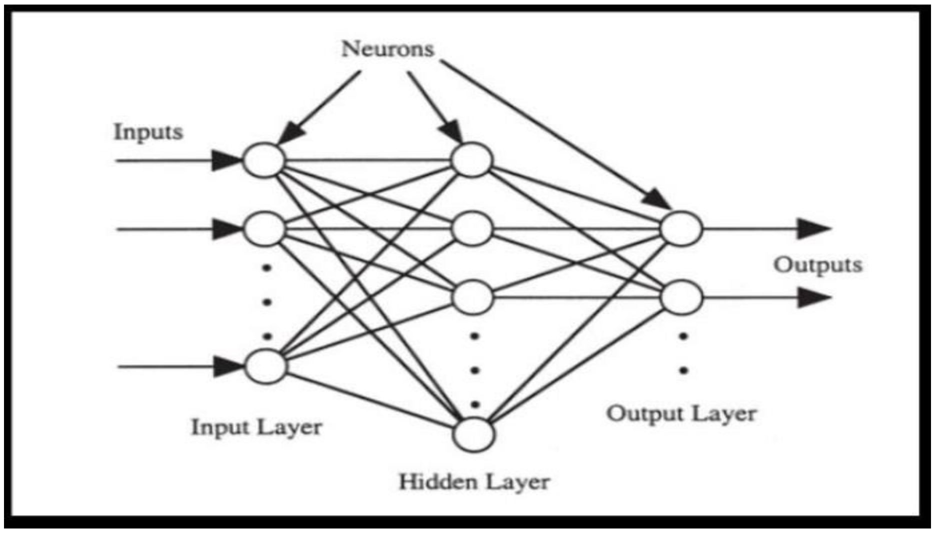


Figure 3

What are Convolutional Neural Networks (CNN)?

A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice. This makes them highly suitable for computer vision (CV) tasks and for applications where object recognition is vital, such as self-driving cars and facial recognition.

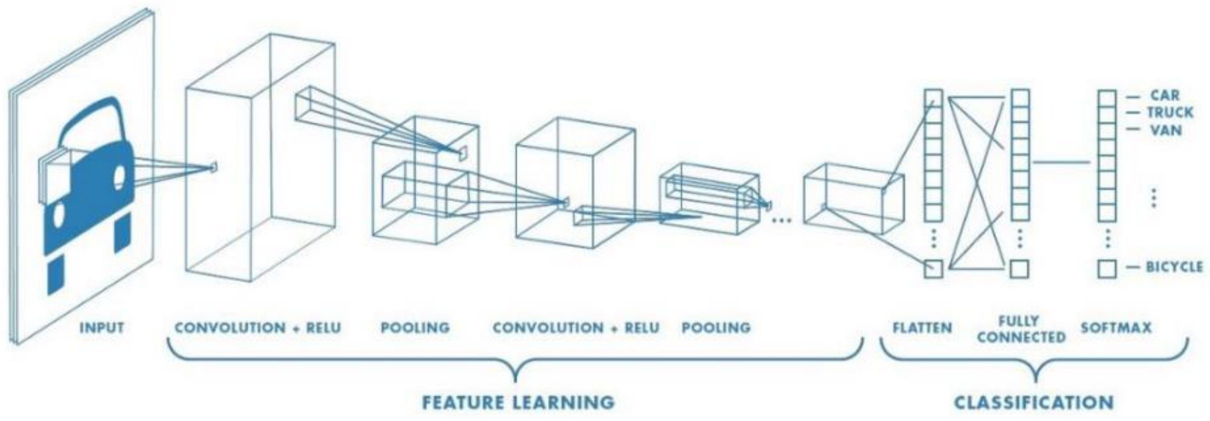


Figure 4

There are two main parts to a CNN architecture

• A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction.

• The network of feature extraction consists of many pairs of convolutional or pooling layers.

• A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

• This CNN model of feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarizes the existing features contained in an original set of features. There are many CNN layers as shown in the CNN architecture diagram.

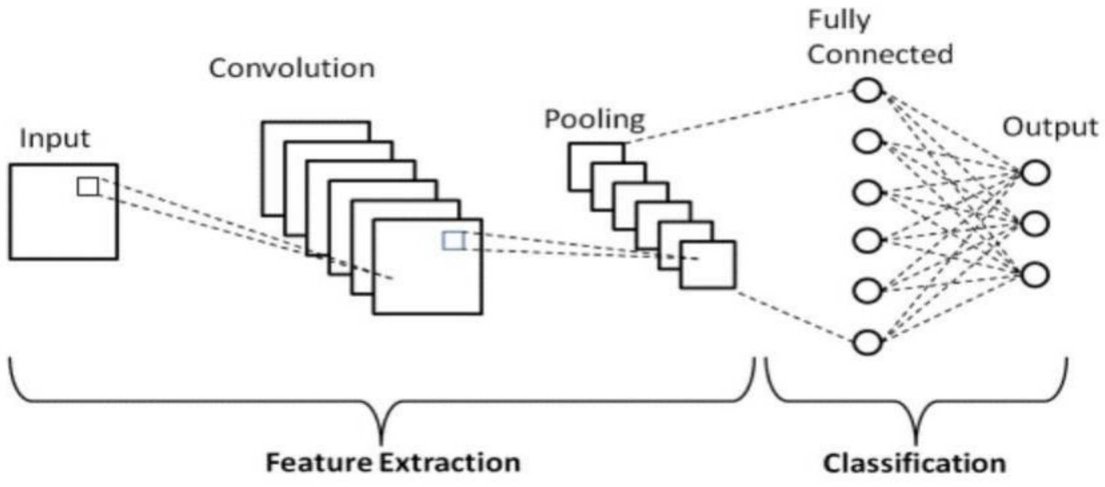


Figure 5

LAYERS OF CNN:

1) **Convolutional Layer**

   This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

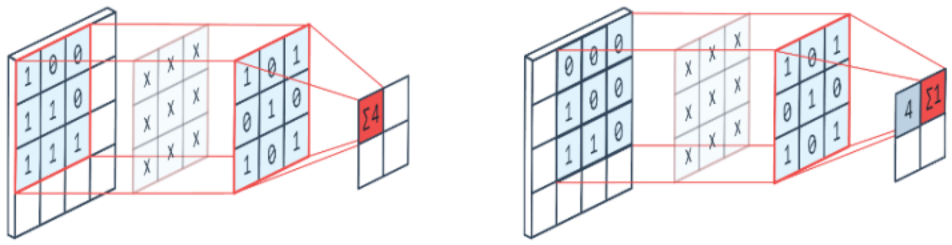


Figure 6

2) **Pooling Layer**

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

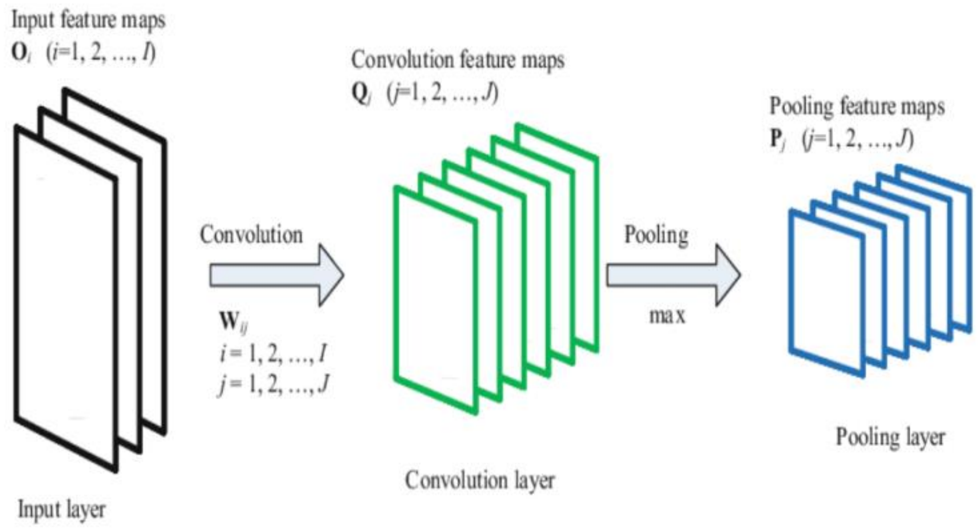


Figure 7

3) **Fully-connected layer**

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture. In this, the input image from the previous layers is flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

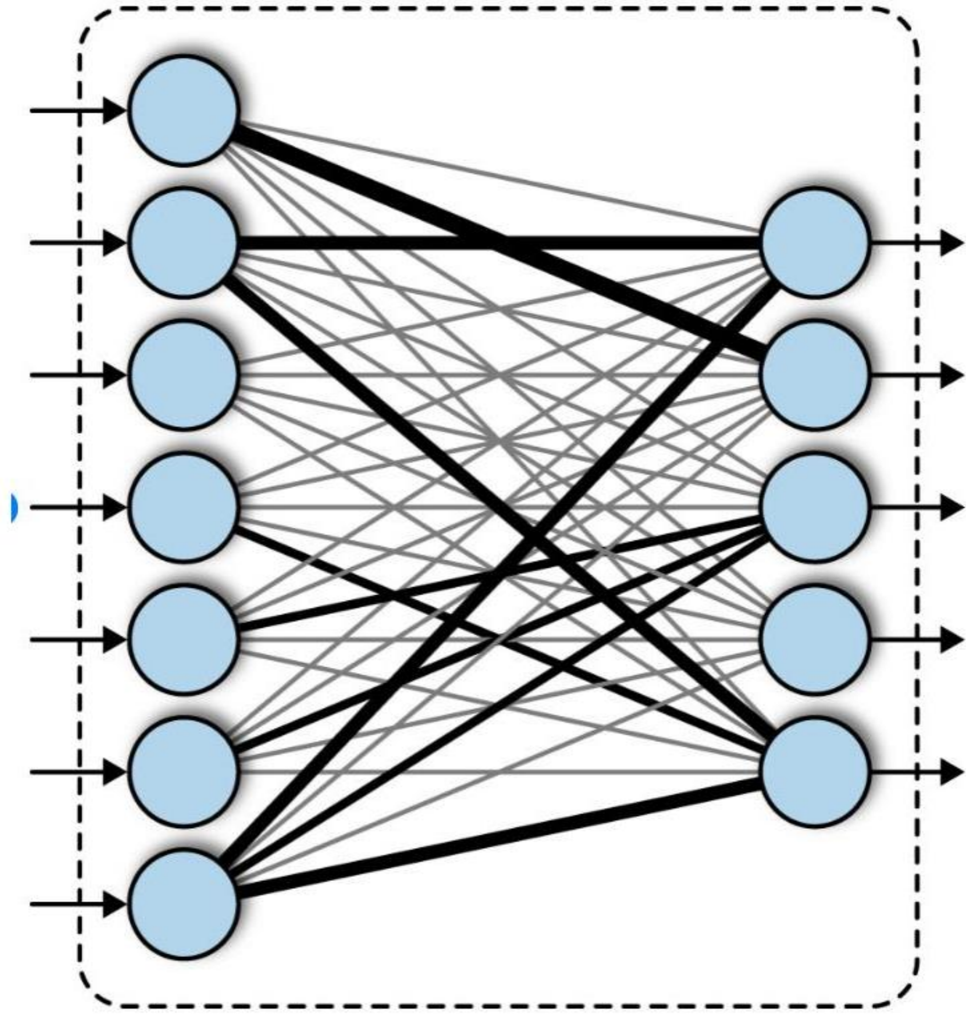


Figure 8

4) **Activation functions**

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.

As mentioned earlier NumPy, Pandas, Tensorflow and matplot is used to represent the accuracy of the model.

Figure 9

Code to implement the project

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
import math
import numpy as np
import pandas as pd

import scikitplot
import seaborn as sns
from matplotlib import pyplot

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report

import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dropout, BatchNormalization, LeakyReLU,
Activation
from tensorflow.keras.callbacks import Callback, EarlyStopping,
ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from keras.utils import np_utils
df = pd.read_csv('../input/facial-expression-
recognitionferchallenge/fer2013/fer2013/fer2013.csv')
```

```
print(df.shape)
df.head()
df.emotion.unique()
emotion_label_to_text = {0:'anger', 1:'disgust', 2:'fear', 3:'happiness', 4: 'sadness', 5:
'surprise', 6: 'neutral'}
df.emotion.value_counts()
sns.countplot(df.emotion)
pyplot.show()
math.sqrt(len(df.pixels[0].split(' ')))
fig = pyplot.figure(1, (14, 14))

k = 0
for label in sorted(df.emotion.unique()):
    for j in range(7):
        px = df[df.emotion==label].pixels.iloc[k]
        px = np.array(px.split(' ')).reshape(48, 48).astype('float32')

        k += 1
        ax = pyplot.subplot(7, 7, k)
        ax.imshow(px, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(emotion_label_to_text[label])
        pyplot.tight_layout()
INTERESTED_LABELS = [3, 4, 6]
df = df[df.emotion.isin(INTERESTED_LABELS)]
df.shape
img_array = df.pixels.apply(lambda x: np.array(x.split(' ')).reshape(48, 48,
1).astype('float32'))
img_array = np.stack(img_array, axis=0)
img_array.shape
le = LabelEncoder()
img_labels = le.fit_transform(df.emotion)
img_labels = np_utils.to_categorical(img_labels)
img_labels.shape
le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print(le_name_mapping)
X_train, X_valid, y_train, y_valid = train_test_split(img_array, img_labels,
                                                    shuffle=True, stratify=img_labels,
                                                    test_size=0.1, random_state=42)
```

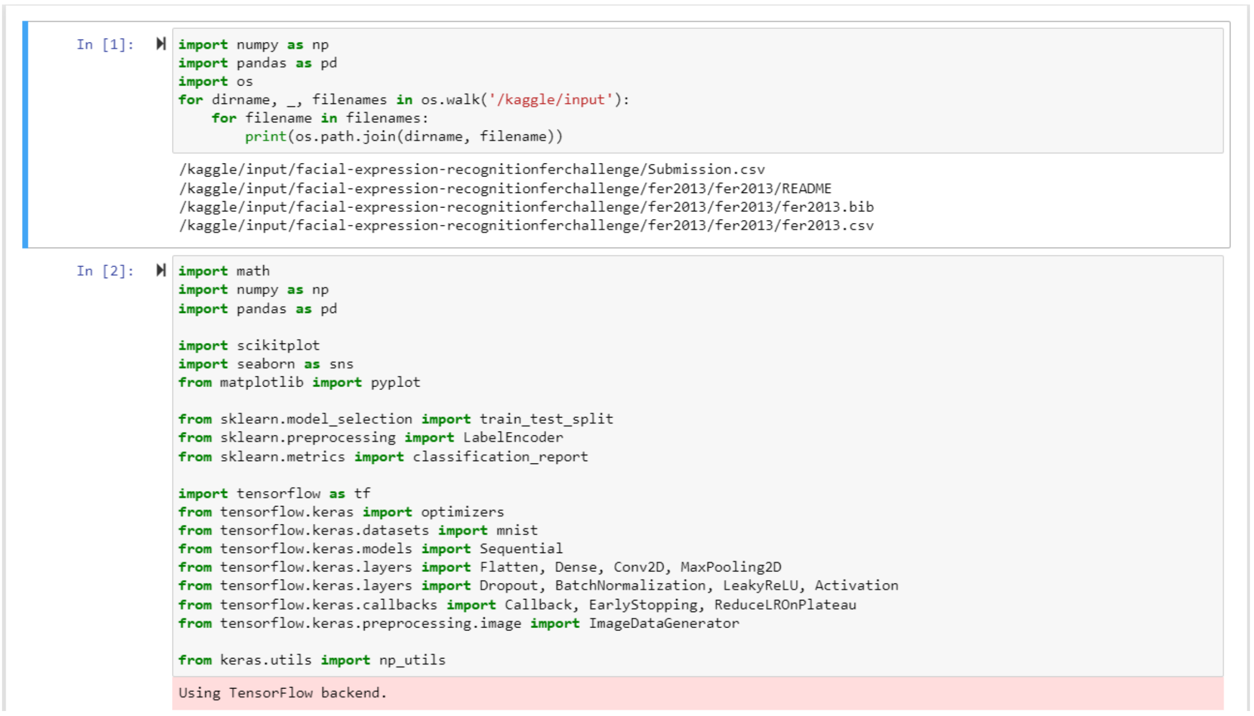
```
X_train.shape, X_valid.shape, y_train.shape, y_valid.sha
del df
del img_array
del img_labels
img_width = X_train.shape[1]
img_height = X_train.shape[2]
img_depth = X_train.shape[3]
num_classes = y_train.shape[1]
X_train = X_train / 255.
X_valid = X_valid / 255.
def build_net(optim):
    """
    This is a Deep Convolutional Neural Network (DCNN). For generalization
purpose I used dropouts in regular intervals.
    I used `ELU` as the activation because it avoids dying relu problem but also
performed well as compared to LeakyRelu
    atleast in this case. `he_normal` kernel initializer is used as it suits ELU.
BatchNormalization is also used for better
    results.
    """
    net = Sequential(name='DCNN')

    net.add(
        Conv2D(
            filters=64,
            kernel_size=(5,5),
            input_shape=(img_width, img_height, img_depth),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_1'
        )
    )
    net.add(BatchNormalization(name='batchnorm_1'))
    net.add(
        Conv2D(
            filters=64,
            kernel_size=(5,5),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_2'
        )
    )
    net.add(BatchNormalization(name='batchnorm_2'))

    net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_1'))
    net.add(Dropout(0.4, name='dropout_1'))

    net.add(
        Conv2D(
            filters=128,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_3'
        )
    )
    net.add(BatchNormalization(name='batchnorm_3'))
    net.add(
        Conv2D(
            filters=128,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_4'
        )
    )
    net.add(BatchNormalization(name='batchnorm_4'))

    net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_2'))
    net.add(Dropout(0.4, name='dropout_2'))

    net.add(
        Conv2D(
            filters=256,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_5'
        )
    )
    net.add(BatchNormalization(name='batchnorm_5'))
    net.add(
        Conv2D(
            filters=256,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_6'
        )
    )
    net.add(BatchNormalization(name='batchnorm_6'))

    net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_3'))
    net.add(Dropout(0.5, name='dropout_3'))

    net.add(Flatten(name='flatten'))

    net.add(
        Dense(
            128,
            activation='elu',
            kernel_initializer='he_normal',
            name='dense_1'
        )
    )
    net.add(BatchNormalization(name='batchnorm_7'))

    net.add(Dropout(0.6, name='dropout_4'))

    net.add(
        Dense(
            num_classes,
            activation='softmax',
            name='out_layer'
        )
    )

    net.compile(
        loss='categorical_crossentropy',
        optimizer=optim,
        metrics=['accuracy']
    )

    net.summary()

    return net
"""
I used two callbacks one is `early stopping` for avoiding overfitting training data
and other `ReduceLROnPlateau` for learning rate.
"""

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    min_delta=0.00005,
    patience=11,
    verbose=1,
    restore_best_weights=True,
)

lr_scheduler = ReduceLROnPlateau(
    monitor='val_accuracy',
    factor=0.5,
    patience=7,
    min_lr=1e-7,
    verbose=1,
)

callbacks = [
    early_stopping,
    lr_scheduler,
]
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.15,
    zoom_range=0.15,
    horizontal_flip=True,
)
train_datagen.fit(X_train)
batch_size = 32 #batch size of 32 performs the best.
epochs = 100
optims = [
    optimizers.Nadam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-
07, name='Nadam'),
    optimizers.Adam(0.001),
]

# I tried both `Nadam` and `Adam`, the difference in results is not different but I
finally went with Nadam as it is more popular.
model = build_net(optims[1])
history = model.fit_generator(
    train_datagen.flow(X_train, y_train, batch_size=batch_size),
    validation_data=(X_valid, y_valid),
    steps_per_epoch=len(X_train) / batch_size,
    epochs=epochs,
    callbacks=callbacks,
    use_multiprocessing=True
model_yaml = model.to_yaml()
with open("model.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)

model.save("model.h5")
sns.set()
fig = pyplot.figure(0, (12, 4))

ax = pyplot.subplot(1, 2, 1)
sns.lineplot(history.epoch, history.history['accuracy'], label='train')
sns.lineplot(history.epoch, history.history['val_accuracy'], label='valid')
pyplot.title('Accuracy')
pyplot.tight_layout()

ax = pyplot.subplot(1, 2, 2)
sns.lineplot(history.epoch, history.history['loss'], label='train')
sns.lineplot(history.epoch, history.history['val_loss'], label='valid')
pyplot.title('Loss')
pyplot.tight_layout()

pyplot.savefig('epoch_history_dcnn.png')
pyplot.show()
df_accu = pd.DataFrame({'train': history.history['accuracy'], 'valid':
history.history['val_accuracy']})
df_loss = pd.DataFrame({'train': history.history['loss'], 'valid':
history.history['val_loss']})

fig = pyplot.figure(0, (14, 4))
ax = pyplot.subplot(1, 2, 1)
sns.violinplot(x="variable", y="value", data=pd.melt(df_accu), showfliers=False)
pyplot.title('Accuracy')
pyplot.tight_layout()

ax = pyplot.subplot(1, 2, 2)
sns.violinplot(x="variable", y="value", data=pd.melt(df_loss), showfliers=False)
pyplot.title('Loss')
pyplot.tight_layout()

pyplot.savefig('performance_dist.png')
pyplot.show()
yhat_valid = model.predict_classes(X_valid)
scikitplot.metrics.plot_confusion_matrix(np.argmax(y_valid, axis=1), yhat_valid,
figsize=(7,7))
pyplot.savefig("confusion_matrix_dcnn.png")

print(f'total wrong validation predictions: {np.sum(np.argmax(y_valid, axis=1) !=
yhat_valid)}\n\n')
print(classification_report(np.argmax(y_valid, axis=1), yhat_valid))
mapper = {
    0: "happy",
    1: "sad",
    2: "neutral",
}
np.random.seed(2)
random_sad_imgs = np.random.choice(np.where(y_valid[:, 1]==1)[0], size=9)
random_neutral_imgs = np.random.choice(np.where(y_valid[:, 2]==1)[0], size=9)
```

```
fig = pyplot.figure(1, (18, 4))

for i, (sadidx, neuidx) in enumerate(zip(random_sad_imgs,
random_neutral_imgs)):
        ax = pyplot.subplot(2, 9, i+1)
        sample_img = X_valid[sadidx,:,:,0]
        ax.imshow(sample_img, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"true:sad,
pred:{mapper[model.predict_classes(sample_img.reshape(1,48,48,1))[0]]}")

        ax = pyplot.subplot(2, 9, i+10)
        sample_img = X_valid[neuidx,:,:,0]
        ax.imshow(sample_img, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"t:neut,
p:{mapper[model.predict_classes(sample_img.reshape(1,48,48,1))[0]]}")

        pyplot.tight_layout()
```

Here model is created and saved with name “model”. It is loaded into an application on a mobile phone to activate the model and then the given input is detected by it.

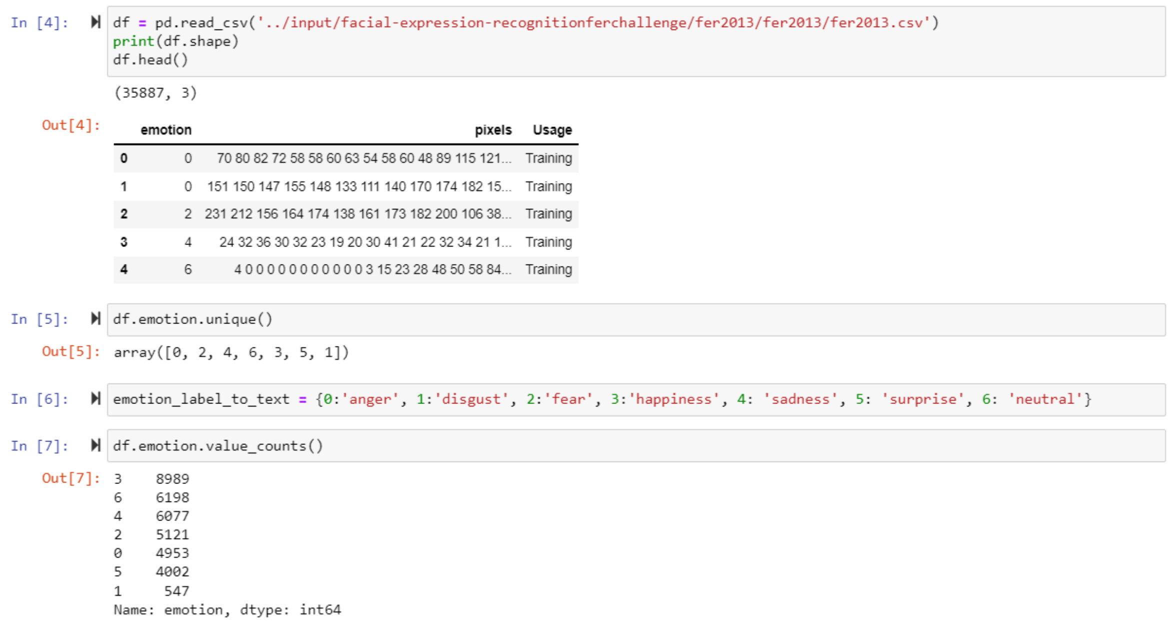
CHAPTER 4.

RESULTS ANALYSIS AND VALIDATION



```
In [1]:  import numpy as np
         import pandas as pd
         import os
         for dirname, _, filenames in os.walk('/kaggle/input'):
             for filename in filenames:
                 print(os.path.join(dirname, filename))

         /kaggle/input/facial-expression-recognitionferchallenge/Submission.csv
         /kaggle/input/facial-expression-recognitionferchallenge/fer2013/fer2013/README
         /kaggle/input/facial-expression-recognitionferchallenge/fer2013/fer2013/fer2013.bib
         /kaggle/input/facial-expression-recognitionferchallenge/fer2013/fer2013/fer2013.csv
```

```
In [2]:  import math
         import numpy as np
         import pandas as pd

         import scikitplot
         import seaborn as sns
         from matplotlib import pyplot

         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import LabelEncoder
         from sklearn.metrics import classification_report

         import tensorflow as tf
         from tensorflow.keras import optimizers
         from tensorflow.keras.datasets import mnist
         from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D
         from tensorflow.keras.layers import Dropout, BatchNormalization, LeakyReLU, Activation
         from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau
         from tensorflow.keras.preprocessing.image import ImageDataGenerator

         from keras.utils import np_utils

         Using TensorFlow backend.
```



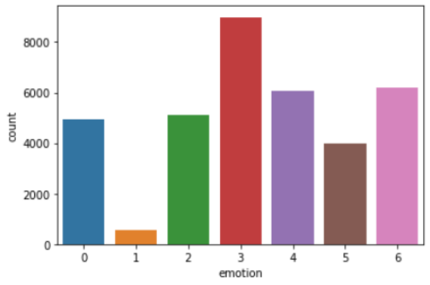
```
In [4]:  df = pd.read_csv('../input/facial-expression-recognitionferchallenge/fer2013/fer2013/fer2013.csv')
         print(df.shape)
         df.head()

         (35887, 3)
```

Out[4]:

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

```
In [5]:  df.emotion.unique()

Out[5]:  array([0, 2, 4, 6, 3, 5, 1])

In [6]:  emotion_label_to_text = {0:'anger', 1:'disgust', 2:'fear', 3:'happiness', 4: 'sadness', 5: 'surprise', 6: 'neutral'}

In [7]:  df.emotion.value_counts()

Out[7]:  3    8989
         6    6198
         4    6077
         2    5121
         0    4953
         5    4002
         1     547
         Name: emotion, dtype: int64
```

```
In [8]:  sns.countplot(df.emotion)
         pyplot.show()
```



```
So majority classes belongs to 3:Happy, 4:Sad and 6:Neutral nd we are also intersted in these three classes only.
```

```
In [9]:  math.sqrt(len(df.pixels[0].split(' ')))

Out[9]:  48.0
```

```
In [10]: fig = pyplot.figure(1, (14, 14))

         k = 0
         for label in sorted(df.emotion.unique()):
             for j in range(7):
                 px = df[df.emotion==label].pixels.iloc[k]
                 px = np.array(px.split(' ')).reshape(48, 48).astype('float32')

                 k += 1
                 ax = pyplot.subplot(7, 7, k)
                 ax.imshow(px, cmap='gray')
                 ax.set_xticks([])
                 ax.set_yticks([])
                 ax.set_title(emotion_label_to_text[label])
                 pyplot.tight_layout()
```

```
In [11]:  INTERESTED_LABELS = [3, 4, 6]
```

```
In [12]:  df = df[df.emotion.isin(INTERESTED_LABELS)]
          df.shape

Out[12]:  (21264, 3)
```

Now I will make the data compatible for neural networks.

```
In [13]:  img_array = df.pixels.apply(lambda x: np.array(x.split(' ')).reshape(48, 48, 1).astype('float32'))
          img_array = np.stack(img_array, axis=0)
```

```
In [14]:  img_array.shape

Out[14]:  (21264, 48, 48, 1)
```

```
In [15]:  le = LabelEncoder()
          img_labels = le.fit_transform(df.emotion)
          img_labels = np_utils.to_categorical(img_labels)
          img_labels.shape

Out[15]:  (21264, 3)
```

```
In [16]:  le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
          print(le_name_mapping)

          {3: 0, 4: 1, 6: 2}
```

Splitting the data into training and validation set.

```
In [17]:  X_train, X_valid, y_train, y_valid = train_test_split(img_array, img_labels,
                                                                shuffle=True, stratify=img_labels,
                                                                test_size=0.1, random_state=42)
          X_train.shape, X_valid.shape, y_train.shape, y_valid.shape

Out[17]:  ((19137, 48, 48, 1), (2127, 48, 48, 1), (19137, 3), (2127, 3))
```

```
In [18]:  del df
          del img_array
          del img_labels
```

```
In [19]:  img_width = X_train.shape[1]
          img_height = X_train.shape[2]
          img_depth = X_train.shape[3]
          num_classes = y_train.shape[1]
```

```
In [20]:  # Normalizing results, as neural networks are very sensitive to unnormalized data.
          X_train = X_train / 255.
          X_valid = X_valid / 255.
```

```
In [21]:  def build_net(optim):
               """
               This is a Deep Convolutional Neural Network (DCNN). For generalization purpose I used dropouts in regular intervals.
               I used `ELU` as the activation because it avoids dying relu problem but also performed well as compared to LeakyRelu
               atleast in this case. `he_normal` kernel initializer is used as it suits ELU. BatchNormalization is also used for better
               results.
               """
               net = Sequential(name='DCNN')

               net.add(
                   Conv2D(
                       filters=64,
                       kernel_size=(5,5),
                       input_shape=(img_width, img_height, img_depth),
                       activation='elu',
                       padding='same',
                       kernel_initializer='he_normal',
                       name='conv2d_1'
                   )
               )
               net.add(BatchNormalization(name='batchnorm_1'))
               net.add(
                   Conv2D(
                       filters=64,
                       kernel_size=(5,5),
                       activation='elu',
                       padding='same',
                       kernel_initializer='he_normal',
                       name='conv2d_2'
                   )
               )
               net.add(BatchNormalization(name='batchnorm_2'))
```

```
    net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_1'))
    net.add(Dropout(0.4, name='dropout_1'))

    net.add(
        Conv2D(
            filters=128,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_3'
        )
    )
    net.add(BatchNormalization(name='batchnorm_3'))
    net.add(
        Conv2D(
            filters=128,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_4'
        )
    )
    net.add(BatchNormalization(name='batchnorm_4'))

    net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_2'))
    net.add(Dropout(0.4, name='dropout_2'))
    net.add(
        Conv2D(
            filters=256,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_5'
        )
    )
    net.add(BatchNormalization(name='batchnorm_5'))
    net.add(
        Conv2D(
            filters=256,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_6'
        )
    )
    net.add(BatchNormalization(name='batchnorm_6'))

    net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_3'))
    net.add(Dropout(0.5, name='dropout_3'))

    net.add(Flatten(name='flatten'))

    net.add(
        Dense(
            128,
            activation='elu',
            kernel_initializer='he_normal',
            name='dense_1'
        )
    )
    net.add(BatchNormalization(name='batchnorm_7'))

    net.add(Dropout(0.6, name='dropout_4'))
```

```
    net.add(
        Dense(
            num_classes,
            activation='softmax',
            name='out_layer'
        )
    )

    net.compile(
        loss='categorical_crossentropy',
        optimizer=optim,
        metrics=['accuracy']
    )

    net.summary()

    return net
```

In [22]:

```
"""
I used two callbacks one is `early stopping` for avoiding overfitting training data
and other `ReduceLROnPlateau` for learning rate.
"""

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    min_delta=0.00005,
    patience=11,
    verbose=1,
    restore_best_weights=True,
)

lr_scheduler = ReduceLROnPlateau(
    monitor='val_accuracy',
    factor=0.5,
    patience=7,
    min_lr=1e-7,
    verbose=1,
)

callbacks = [
    early_stopping,
    lr_scheduler,
]
```

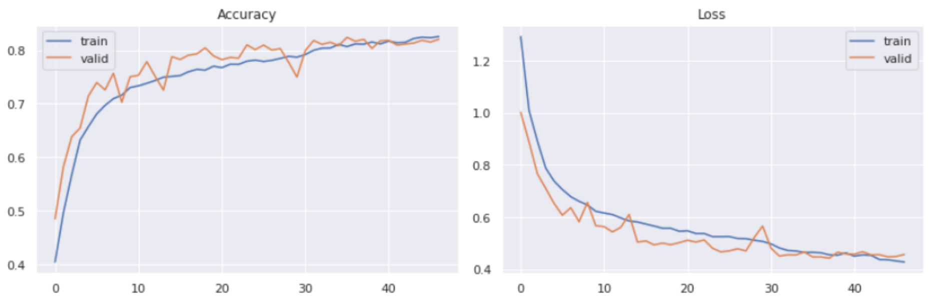
In [23]:
```
# As the data in hand is less as compared to the task so ImageDataGenerator is good to go.
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.15,
    zoom_range=0.15,
    horizontal_flip=True,
)
train_datagen.fit(X_train)
```

In [24]:
```
batch_size = 32 #batch size of 32 performs the best.
epochs = 100
optims = [
    optimizers.Nadam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, name='Nadam'),
    optimizers.Adam(0.001),
]

# I tried both `Nadam` and `Adam`, the difference in results is not different but I finally went with Nadam as it is more pop
model = build_net(optims[1])
history = model.fit_generator(
    train_datagen.flow(X_train, y_train, batch_size=batch_size),
    validation_data=(X_valid, y_valid),
    steps_per_epoch=len(X_train) / batch_size,
    epochs=epochs,
    callbacks=callbacks,
    use_multiprocessing=True
)
```

```
598/598 [============================>.] - ETA: 0s - loss: 0.4552 - accuracy: 0.8148
Epoch 00043: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
599/598 [==============================] - 12s 20ms/step - loss: 0.4552 - accuracy: 0.8148 - val_loss: 0.4559 - val_accur
acy: 0.8115
Epoch 44/100
599/598 [==============================] - 11s 19ms/step - loss: 0.4400 - accuracy: 0.8219 - val_loss: 0.4567 - val_accur
acy: 0.8134
Epoch 45/100
599/598 [==============================] - 11s 19ms/step - loss: 0.4386 - accuracy: 0.8245 - val_loss: 0.4483 - val_accur
acy: 0.8185
Epoch 46/100
599/598 [==============================] - 12s 20ms/step - loss: 0.4336 - accuracy: 0.8235 - val_loss: 0.4497 - val_accur
acy: 0.8152
Epoch 47/100
597/598 [============================>.] - ETA: 0s - loss: 0.4296 - accuracy: 0.8255Restoring model weights from the end
of the best epoch.
599/598 [==============================] - 11s 19ms/step - loss: 0.4295 - accuracy: 0.8256 - val_loss: 0.4576 - val_accur
acy: 0.8204
Epoch 00047: early stopping
```

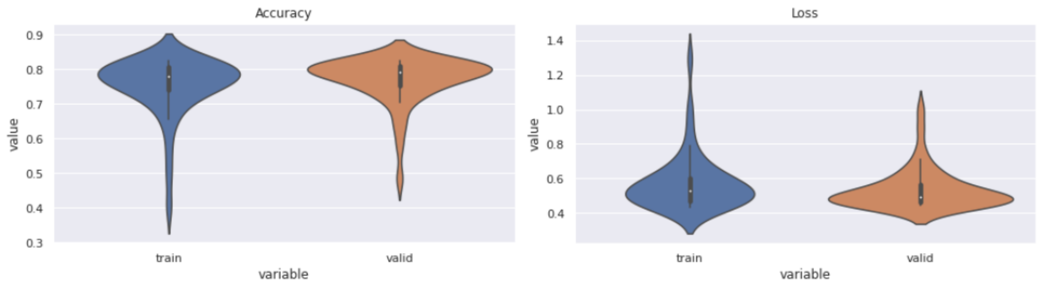
In [25]:
```
model_yaml = model.to_yaml()
with open("model.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)

model.save("model.h5")
```

In [26]:
```
sns.set()
fig = pyplot.figure(0, (12, 4))

ax = pyplot.subplot(1, 2, 1)
sns.lineplot(history.epoch, history.history['accuracy'], label='train')
sns.lineplot(history.epoch, history.history['val_accuracy'], label='valid')
pyplot.title('Accuracy')
pyplot.tight_layout()

ax = pyplot.subplot(1, 2, 2)
sns.lineplot(history.epoch, history.history['loss'], label='train')
sns.lineplot(history.epoch, history.history['val_loss'], label='valid')
pyplot.title('Loss')
pyplot.tight_layout()

pyplot.savefig('epoch_history_dcnn.png')
pyplot.show()
```



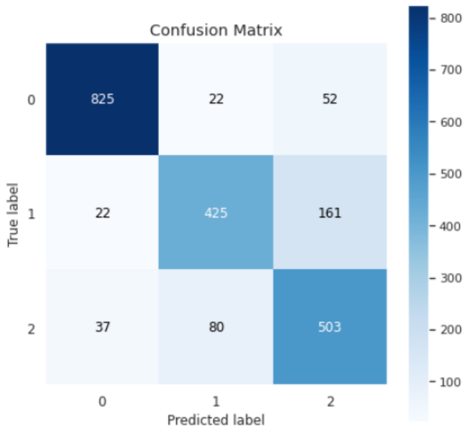
The epochs history shows that accuracy gradually increases and achieved +83% accuracy on both training and validation set, but at the end the model starts overfitting training data.

In [27]:
```
df_accu = pd.DataFrame({'train': history.history['accuracy'], 'valid': history.history['val_accuracy']})
df_loss = pd.DataFrame({'train': history.history['loss'], 'valid': history.history['val_loss']})

fig = pyplot.figure(0, (14, 4))
ax = pyplot.subplot(1, 2, 1)
sns.violinplot(x="variable", y="value", data=pd.melt(df_accu), showfliers=False)
pyplot.title('Accuracy')
pyplot.tight_layout()

ax = pyplot.subplot(1, 2, 2)
sns.violinplot(x="variable", y="value", data=pd.melt(df_loss), showfliers=False)
pyplot.title('Loss')
pyplot.tight_layout()

pyplot.savefig('performance_dist.png')
pyplot.show()
```



In [28]:
```
yhat_valid = model.predict_classes(X_valid)
scikitplot.metrics.plot_confusion_matrix(np.argmax(y_valid, axis=1), yhat_valid, figsize=(7,7))
pyplot.savefig("confusion_matrix_dcnn.png")

print(f'total wrong validation predictions: {np.sum(np.argmax(y_valid, axis=1) != yhat_valid)}\n\n')
print(classification_report(np.argmax(y_valid, axis=1), yhat_valid))
```

```
total wrong validation predictions: 374


              precision    recall  f1-score   support

           0       0.93      0.92      0.93       899
           1       0.81      0.70      0.75       608
           2       0.70      0.81      0.75       620

    accuracy                           0.82      2127
   macro avg       0.81      0.81      0.81      2127
weighted avg       0.83      0.82      0.82      2127
```

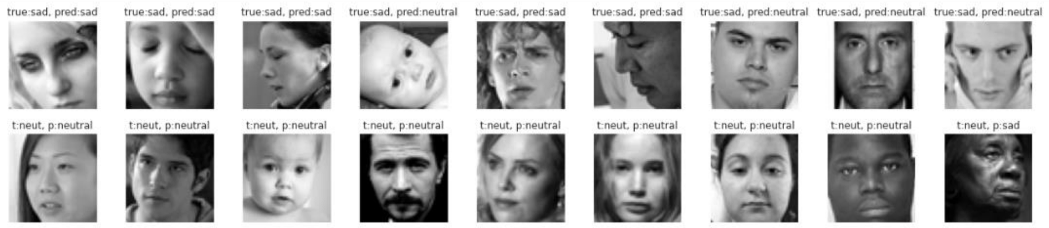


	0	1	2
0	825	22	52
1	22	425	161
2	37	80	503

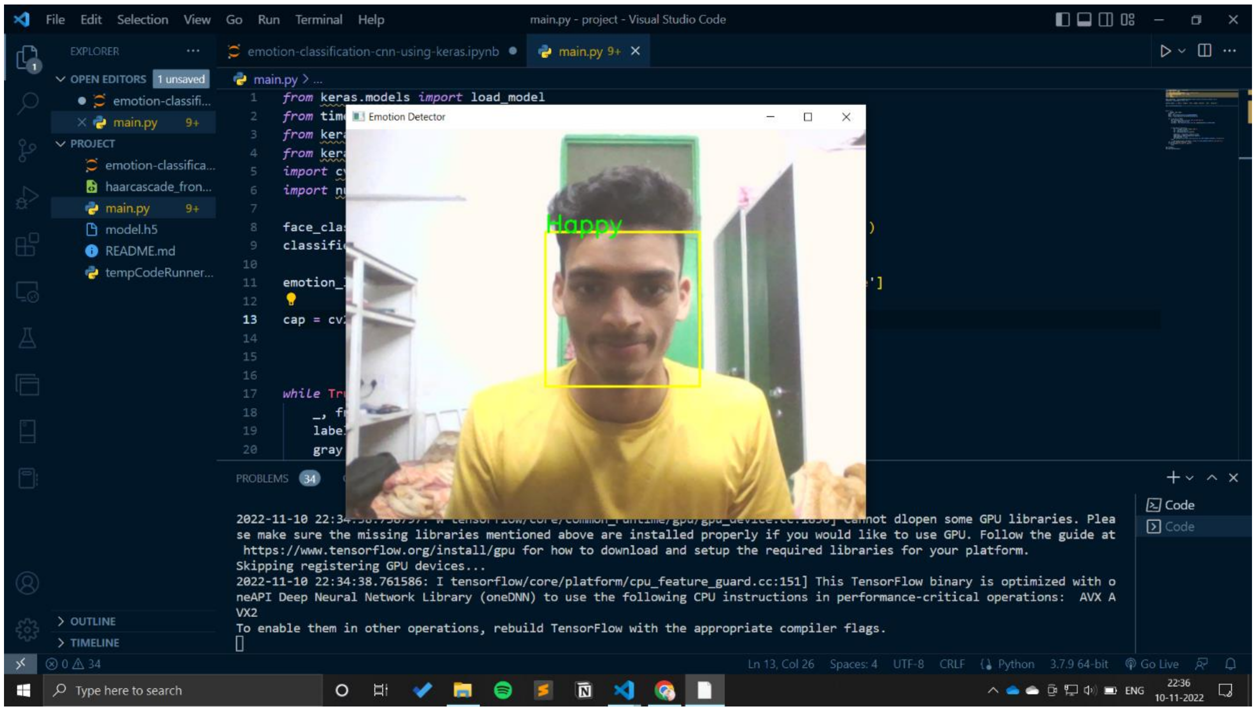
The confusion matrix clearly shows that our model is doing good job on the class `happy` but it's performance is low on other two classes. One of the reason for this could be the fact that these two classes have less data. But when I looked at the images I found some images from these two classes are even hard for a human to tell whether the person is sad or neutral. Facial expression depends on individual as well. Some person's neutral face looks like sad.

In [29]:
```
mapper = {
    0: "happy",
    1: "sad",
    2: "neutral",
}
```

In [30]:
```
np.random.seed(2)
random_sad_imgs = np.random.choice(np.where(y_valid[:, 1]==1)[0], size=9)
random_neutral_imgs = np.random.choice(np.where(y_valid[:, 2]==1)[0], size=9)

fig = pyplot.figure(1, (18, 4))

for i, (sadidx, neuidx) in enumerate(zip(random_sad_imgs, random_neutral_imgs)):
        ax = pyplot.subplot(2, 9, i+1)
        sample_img = X_valid[sadidx,:,:,0]
        ax.imshow(sample_img, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"true:sad, pred:{mapper[model.predict_classes(sample_img.reshape(1,48,48,1))[0]]}")

        ax = pyplot.subplot(2, 9, i+10)
        sample_img = X_valid[neuidx,:,:,0]
        ax.imshow(sample_img, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"t:neut, p:{mapper[model.predict_classes(sample_img.reshape(1,48,48,1))[0]]}")

        pyplot.tight_layout()
```

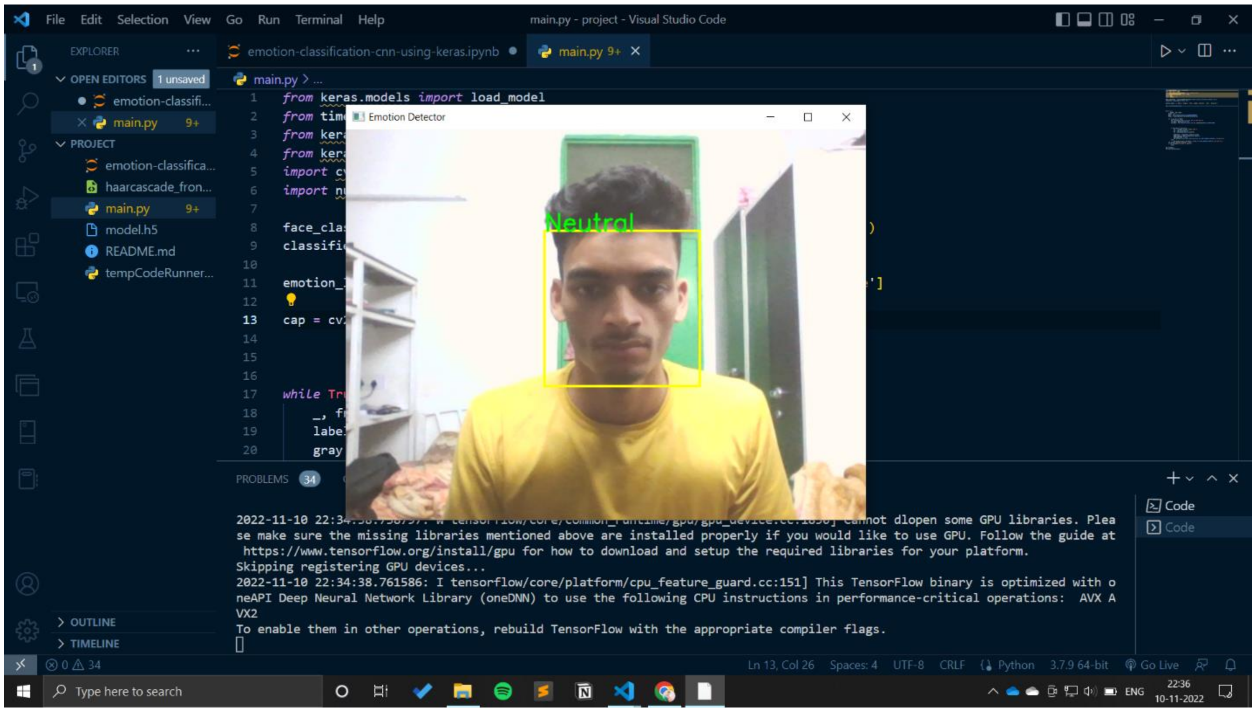


See in the first row 7th image looks more like neutral rather than sad and our model even predicted it neutral. Whereas the last image in second row is very much sad.



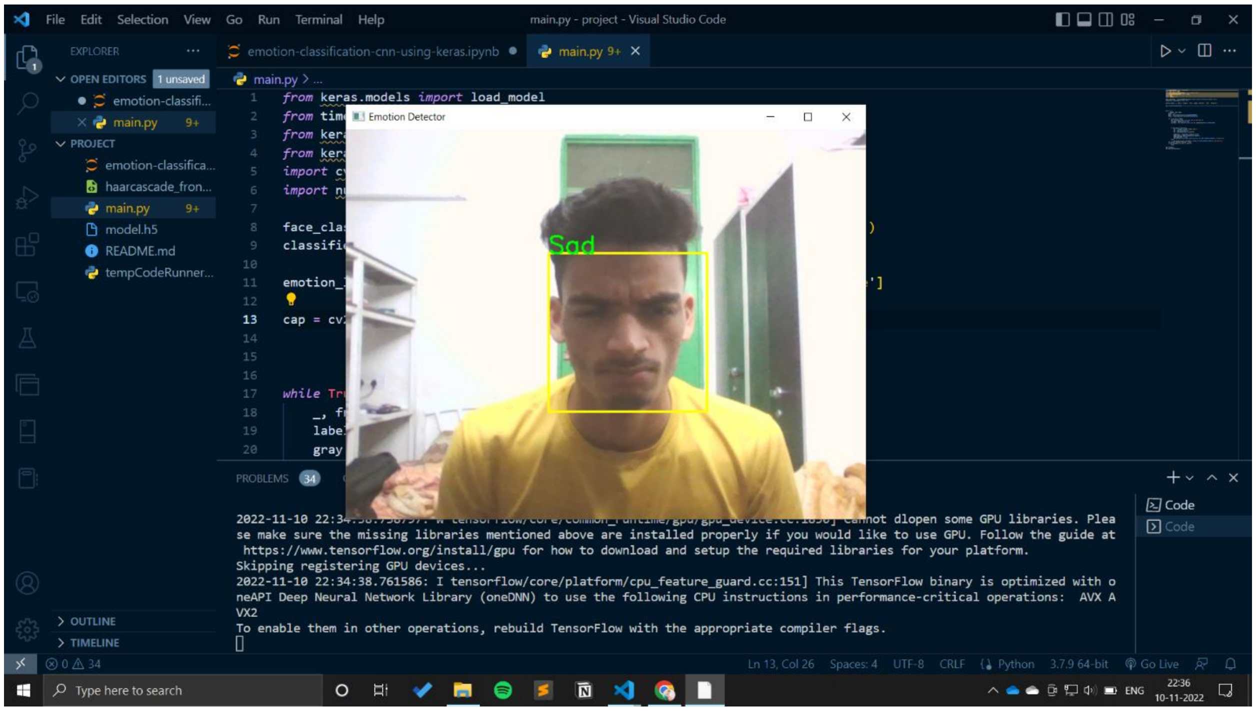
```
from keras.models import load_model
from tim
from ker
from ker
import c
import n

face_cla                                   )
classifi

emotion_                                   ']

cap = cv



while Tr
    _, f
    labe
    gray
```

```
2022-11-10 22:34                                                    nnot dlopen some GPU libraries. Plea
se make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at
 https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2022-11-10 22:34:38.761586: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with o
neAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX A
VX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```



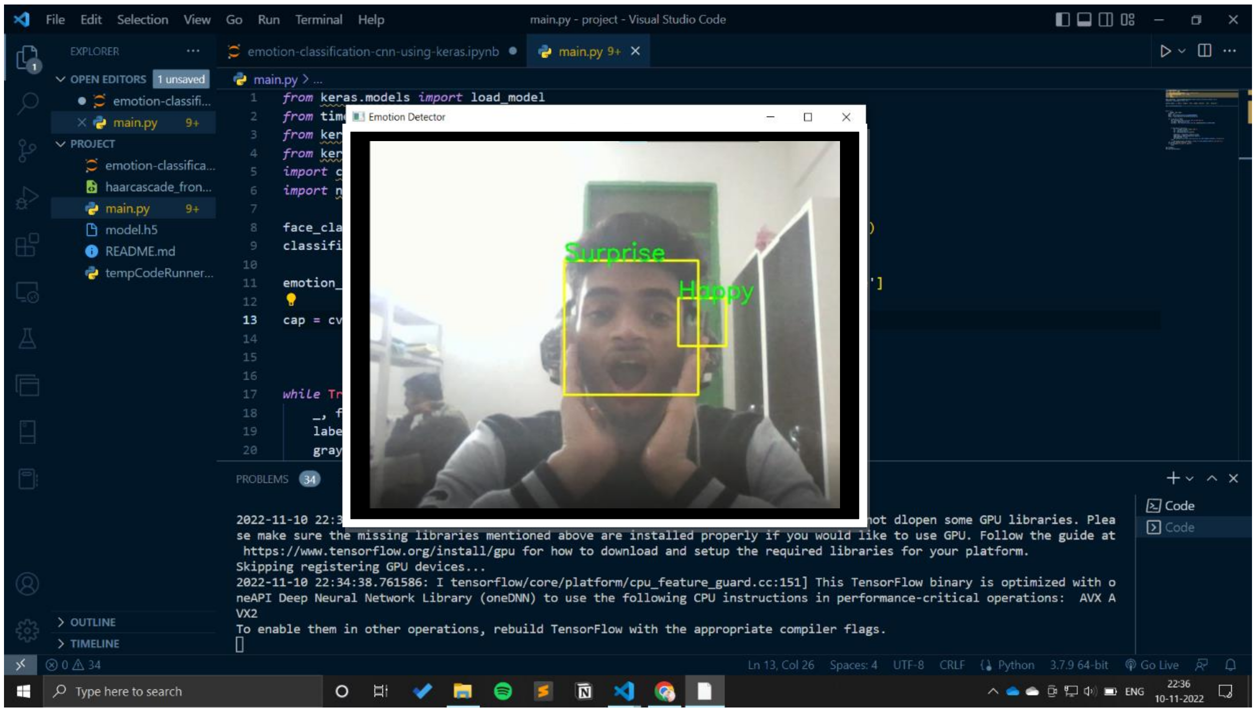
```
from keras.models import load_model
from tim
from ker
from ker
import c
import n

face_cla                                   )
classifi

emotion_                                   ']

cap = cv



while Tr
    _, f
    labe
    gray
```

```
2022-11-10 22:34                                                    nnot dlopen some GPU libraries. Plea
se make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at
 https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2022-11-10 22:34:38.761586: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with o
neAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX A
VX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```



```
from keras.models import load_model
from tim
from ker
from ker
import c
import n

face_cla
classifi

emotion_

cap = cv



while Tr
    _, f
    labe
    gray
```

```
2022-11-10 22:34                                                      nnot dlopen some GPU libraries. Plea
se make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at
 https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2022-11-10 22:34:38.761586: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with o
neAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX A
VX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```



```
from keras.models import load_model
from tim
from ker
from ker
import c
import n

face_cla
classifi

emotion_

cap = cv



while Tr
    _, f
    labe
    gray
```

```
2022-11-10 22:3                                                       not dlopen some GPU libraries. Plea
se make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at
 https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2022-11-10 22:34:38.761586: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with o
neAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX A
VX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

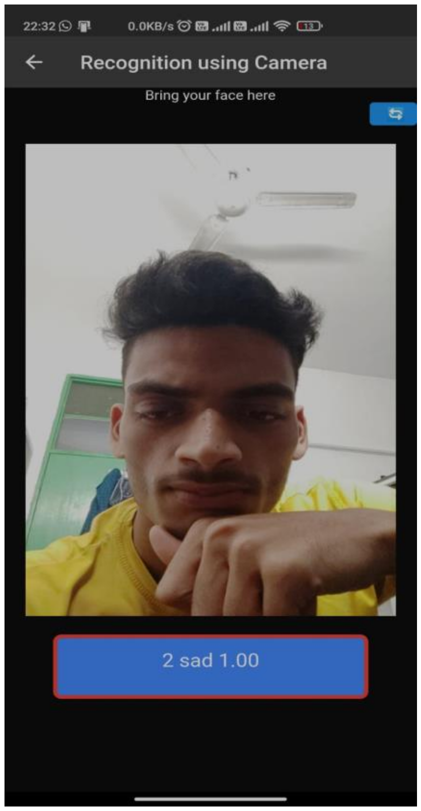

22:32
0.0KB/s
Recognition using Camera
Bring your face here
2 sad 1.00

Figure 10

CHAPTER 5.
CONCLUSION AND FUTURE WORK

5.1. Conclusion

In this project emotions of a human face are being detected efficiently and analyzed the patterns of various kinds of expressions of emotions. Research suggests that the facial expressions accompanying anger, disgust, fear, happiness, sadness, and surprise are human universals, and the amygdala is central to emotion recognition.

5.2. Future work

Technologies based on facial expression recognition (FER, also known as affect recognition) form a significant part of the emotion recognition market, estimated to reach a value of $56 billion by 2024. Emotion recognition provides benefits to many institutions and aspects of life. It is useful and important for security and healthcare purposes. Also, it is crucial for easy and simple detection of human feelings at a specific moment without asking them.

REFERENCES

1) https://www.researchgate.net/publication/344331972_Facial_Emotion_Detection_
Using_Neural_Network/link/5f68e384a6fdcc0086340933/download
2) file:///C:/Users/vicky/Downloads/Facial-Emotion-Detection-Using-NeuralNetwork.pdf
3) https://www.ripublication.com/ijaer18/ijaerv13n8_119.pdf
4) https://peltarion.com/knowledge-center/modeling-view/build-an-aimodel/blocks/2d-
convolution
5) https://www.upgrad.com/blog/basic-cnn-architecture/
6) https://www.researchgate.net/figure/Visualization-of-a-fully-connected-layerTaken-from-
Hollemans-72_fig4_336607800
7) https://www.researchgate.net/figure/A-pair-of-convolution-and-pooling-layers-inthe-CNN-
architecture_fig1_323233791
8) https://en.wikipedia.org/wiki/NumPy
9) https://en.wikipedia.org/wiki/Pandas_(software)
10) https://en.wikipedia.org/wiki/TensorFlow
11) https://matplotlib.org/stable/gallery/misc/logos2.html