CHAPTER : 7 | IMPLEMENTATION

7.1 *Implementation*……

[1]:
```
# Importing the Dependencies

from keras.utils import to_categorical
from keras.preprocessing.image import load_img
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
from tqdm.notebook import tqdm
from sklearn.preprocessing import LabelEncoder
import os
import pandas as pd
import numpy as np
```

Fig 2 : All the Dependencies Used in The Project

[2]:
```
# Function to make a data frame through directories

def createdataframe(dir):
    image_paths = []
    labels = []
    for label in os.listdir(dir):
        for imagename in os.listdir(os.path.join(dir, label)):
            image_paths.append(os.path.join(dir, label, imagename))
            labels.append(label)
        print(label, "completed")
    return image_paths, labels
```

[3]:
```
# Making test and train directories

Train_DIR = 'D:/Projects/Emotion Detection System/images/train'
Test_DIR = 'D:/Projects/Emotion Detection System/images/test'
```

Fig 3 : Function to make data frame from the DIR dataset

[6]:
```
# Making a function to extract all the features of the image

def extract_features(images):
    features = []
    for image in tqdm(images):
        img = load_img(image, color_mode = 'grayscale')
        img = np.array(img)
        features.append(img)
    features = np.array(features)
    features = features.reshape(len(features), 48, 48, 1)
    return features
```

[7]:
```
# Extracting the features of the image

train_features = extract_features(train['image'])
```

Error displaying widget: model not found

[8]:
```
# Extracting the features of the image

test_features = extract_features(test['image'])
```

Error displaying widget: model not found

Fig 4 : Function to Extract all the features of data

[9]:
```
# Dividing each feature with pixel size to make X_Train and X_Test

X_Train = train_features/255.0
X_Test = test_features/255.0
```

[10]:
```
# Making Y_Train and Y_Test

le = LabelEncoder()
le.fit(train['label'])
```

[10]:
```
LabelEncoder()
```

[11]:
```
# Making Y_Train and Y_Test

Y_Train = le.transform(train['label'])
Y_Test = le.transform(test['label'])
```

[12]:
```
Y_Train = to_categorical(Y_Train, num_classes = 7)
Y_Test = to_categorical(Y_Test, num_classes = 7)
```

Fig 5 : Making X_Train, X_Test and Y_Train , Y_Test

[13]:
```
# Making Convolutionl Neural Network

model = Sequential()

#convolutional layers
model.add(Conv2D(128, kernel_size = (3,3), activation = 'relu', input_shape = (48,48,1)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(256, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(512, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(512, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Flatten())

#fully connected layers
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.3))

#output layer
model.add(Dense(7, activation = 'softmax'))
```

Fig 6 : Convolutional Neural Network Using Sequential Model

[14]:
```
# Compiling the model

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[18]:
```
# Training the model

model.fit(x = X_Train, y = Y_Train, batch_size = 128, epochs = 300, validation_data = (X_Test, Y_Test))
```

```
Epoch 293/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 949ms/step - accuracy: 0.8663 - loss: 0.4087 - val_accuracy: 0.6269 - val_loss: 1.1789
Epoch 294/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 946ms/step - accuracy: 0.8645 - loss: 0.3996 - val_accuracy: 0.6318 - val_loss: 1.2057
Epoch 295/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 947ms/step - accuracy: 0.8636 - loss: 0.4001 - val_accuracy: 0.6316 - val_loss: 1.2334
Epoch 296/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 948ms/step - accuracy: 0.8673 - loss: 0.3996 - val_accuracy: 0.6291 - val_loss: 1.1961
Epoch 297/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 947ms/step - accuracy: 0.8602 - loss: 0.4256 - val_accuracy: 0.6238 - val_loss: 1.2144
Epoch 298/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 948ms/step - accuracy: 0.8728 - loss: 0.3935 - val_accuracy: 0.6272 - val_loss: 1.2415
Epoch 299/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 948ms/step - accuracy: 0.8687 - loss: 0.4002 - val_accuracy: 0.6323 - val_loss: 1.2533
Epoch 300/300
226/226 ━━━━━━━━━━━━━━━━━━━━ 214s 948ms/step - accuracy: 0.8683 - loss: 0.4006 - val_accuracy: 0.6303 - val_loss: 1.2547
```

[18]:
```
<keras.src.callbacks.history.History at 0x20e0a4b2710>
```

Fig 7 : Training and Accuracy score of CNNN Model

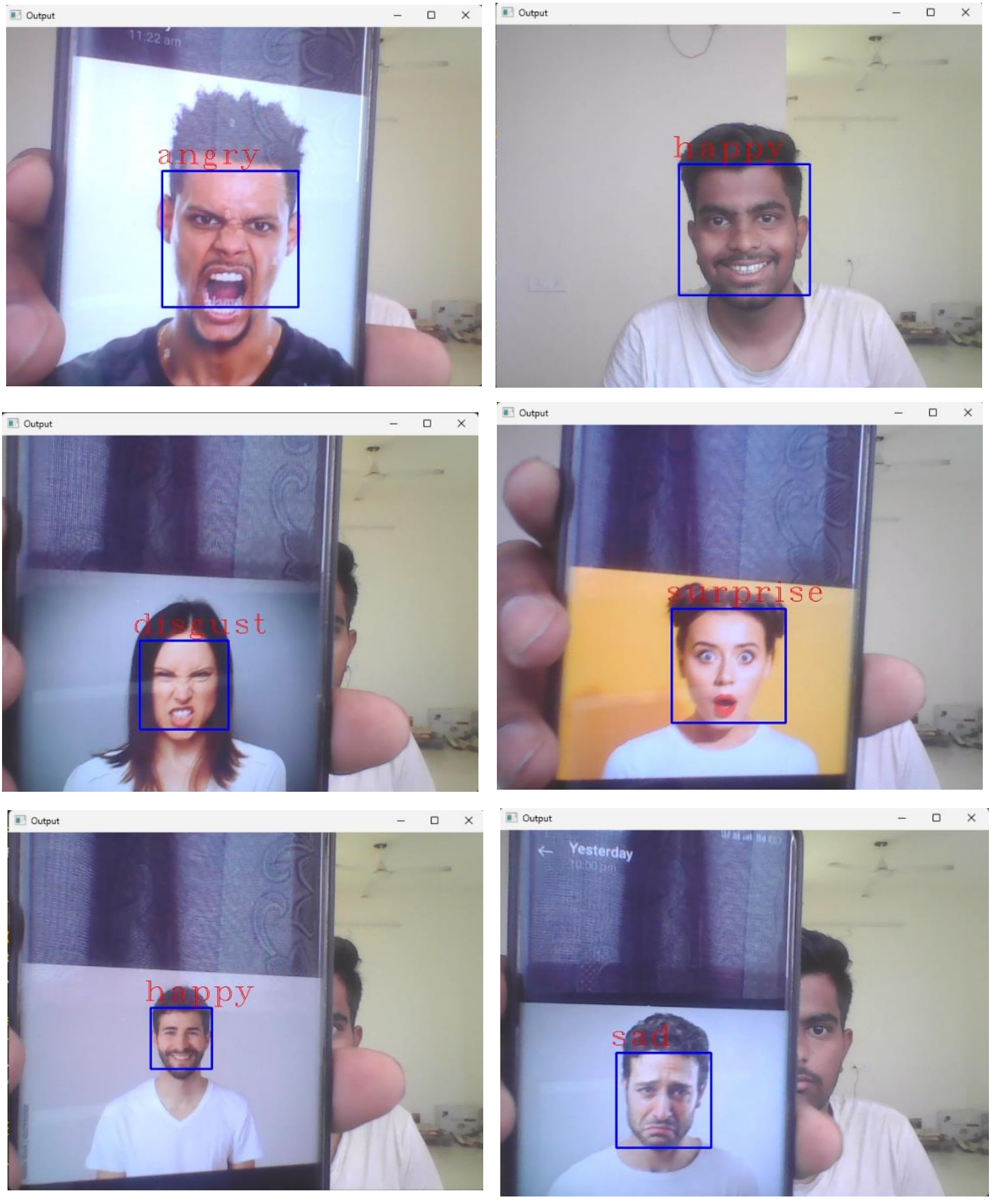

Fig 8 : Outputs