ABSTRACT

Facial expressions are the visible manifestation of the affective state, cognitive activity, intention, personality and psychopathology of a person and plays a communicative role in interpersonal relations. Automatic recognition of facial expressions can be an important component of natural human-machine interfaces; it may also be used in behavioural science and in clinical practice. An automatic Facial Expression Recognition system needs to perform detection and location of faces in a cluttered scene, facial feature extraction, and facial expression classification. Facial expression recognition system is implemented using Convolution Neural Network (CNN). CNN model of the project is based on LeNet Architecture. Kaggle facial expression dataset with seven facial expression labels as happy, sad, surprise, fear, anger, disgust, and neutral is used in this project.

Keywords: ***Facial Expression Recognition, Convolution, CNN, LeNet, Kaggle***

.

LIST OF FIGURES

LIST OF TABLES

Contents

CHAPTER 1

INTRODUCTION

1.1 Motivation

Huge investments are made by large corporations in feedbacks and surveys but they still fail to get equitable response on their investments. The motivation behind choosing this topic is drawn from this situation. An Automatic Facial Expression Recognition is a technology that can prove to be a great aid in improving product and services performance by monitoring customer facial expressions to certain products or service staff by their evaluation.

1.2 Objectives of the Project

- Our aim is to propose an approach that can achieve the desired goal of facial expression detection and identification effectively.

- Facial expression recognition system will be implemented using Convolution Neural Network (CNN) to classify images of human faces into discrete expression categories and compare the result of different datasets.

- CNN model of the project is based on LeNet Architecture.

.

CHAPTER 2

LITERATURE SURVEY

The project makes use of the whole frontal face image and processes it in order to end up with the classifications of 7 universal facial expression prototypes: disgust, fear, joy, surprise, sadness, anger and neutral, using CNN based LeNet Architecture[***Convolutional Neural Networks (LeNet) – Deep Learning 0.1 documentation". Deep Learning 0.1. LISA Lab. Retrieved 31 August***]
Two different approaches have been studied for facial expression recognition. In both of these approaches, two different methodologies, namely the 'Geometric based' and the 'Appearance-based' parameterizations, can be used.

Making use of the whole frontal face image and processing it in order to end up with the classifications of 7 universal facial expression prototypes: disgust, fear, joy, surprise, sadness , neutral and anger; outlines the first approach[***Robust facial expression recognition using local binary patterns. In Image Processing, 2005. ICIP 2005. IEEE International Conference on Vol. 2, pp. II-370). IEEE***.]. Here, it is assumed that each of the above- mentioned emotions have characteristic expressions on face and that's why recognition of them is necessary and sufficient.

Instead of using the face images as a whole, dividing them into some subsections for further processing forms up the main idea of the second approach for facial expression analysis.[***Chibelushi, C. C., & Bourel, F. (2003). Facial expression recognition: A brief tutorial overview. CVonline: On-Line Compendium of Computer Vision, 9***] As expression is more related with subtle changes of some discrete features such as eyes, eyebrows and lip corners; these fine-grained changes are used for analysing automated

recognition. Different features such as Gabor, Haar wavelet coefficients, together with feature extraction and selection methods such as PCA, LDA, and Adaboost are used within this framework.

CHAPTER 3

METHODOLOGY

Detailed Methodology

The facial expression recognition system is implemented using convolutional neural network. The block diagram of the system is shown in following figures:

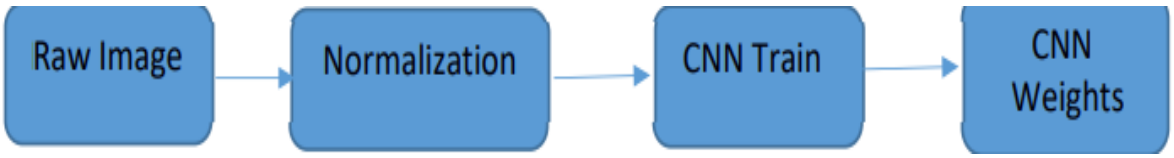

Fig 1: Training Phase

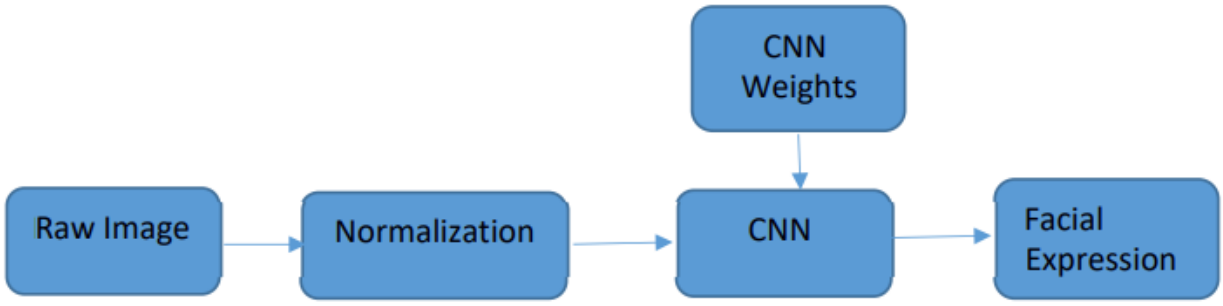

Fig 2: Testing Phase

During training, the system received a training data comprising grayscale images of faces with their respective expression label and learns a set of weights for the network. The training step took as input an image with a face. Thereafter, an intensity normalization is applied to

the image. The normalized images are used to train the Convolutional Network. To ensure that the training performance is not affected by the order of presentation of the examples, validation dataset is used to choose the final best set of weights out of a set of trainings performed with samples presented in different orders. The output of the training step is a set of weights that achieve the best result with the training data. During test, the system received a grayscale image of a face from test dataset, and output the predicted expression by using the final network weights learned during training. Its output is a single number that represents one of the seven basic expressions

3.1 Dataset

The dataset from a Kaggle Facial Expression Recognition Challenge (FER2013) is used for the training and testing. It comprises pre-cropped, 48-by-48-pixel grayscale images of faces each labelled with one of the 7 emotion classes: anger, disgust, fear, happiness, sadness, surprise, and neutral.

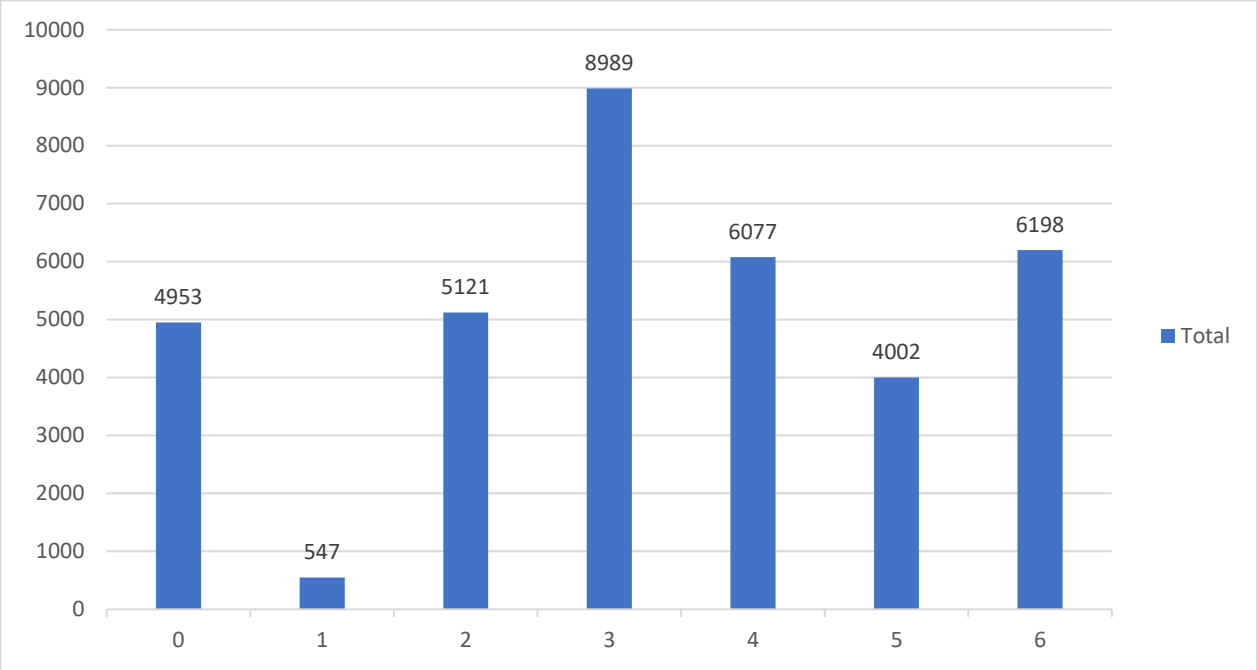

Figure 3: Graphical Representation of dataset distribution

Emotion	Emoticon Meaning	Count of emotion
0	Anger	4953
1	Disgust	547
2	Fear	5121
3	Happiness	8989
4	Sadness	6077
5	Surprise	4002
6	Neutral	6198
Grand Total		**35887**

Table 1: Dataset Distribution

3.2 Architecture of CNN

A typical architecture of a convolutional neural network contains an input layer, some convolutional layers, some fully-connected layers, and an output layer. CNN is designed with some modification on LeNet Architecture .It has 6 layers without considering input and output.

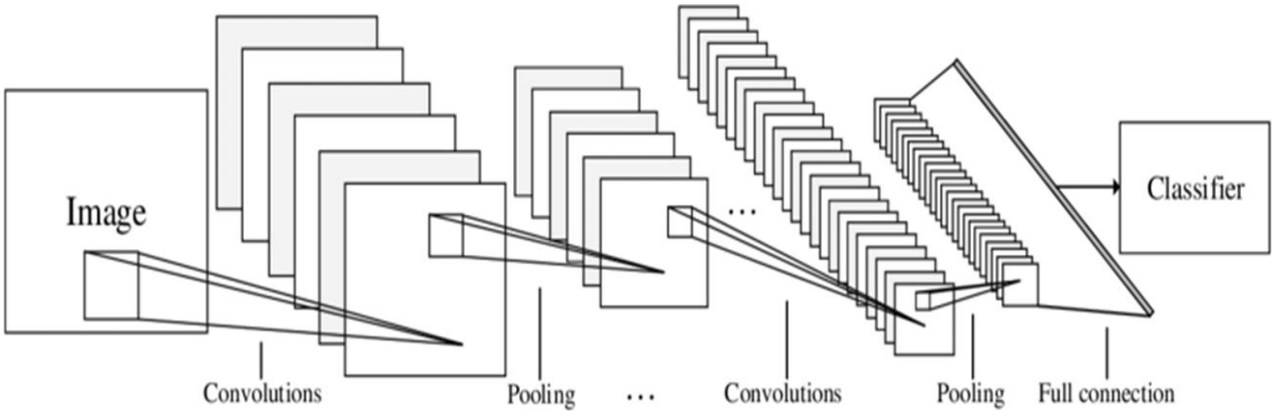

Fig 4: LeNet Architecture

3.2.1. Input Layer

The input layer has pre-determined, fixed dimensions, so the image must be pre-processed before it can be fed into the layer. Normalized Gray scale images of size 48 X 48 pixels from Kaggle dataset are used for training, validation and testing. For testing propose laptop webcam images are also used, in which face is detected and cropped using OpenCV Haar Cascade Classifier and normalized.

3.2.2. Convolution and Pooling Layer

Convolution and pooling are done based on batch processing. Each batch has N images and CNN filter weights are updated on those batches. In each convolution layer, four-dimensional convolution is calculated

between image batch and feature maps. After convolution only, parameters that change are image width and height.

New image width = old image width – filter width + 1

New image height = old image height – filter height +1

After each convolution layer down sampling / subsampling is done for dimensionality reduction. This process is called Pooling. In this project Maxpooling is used.

3.2.3.Fully connected Layer

This layer is inspired by the way neurons transmit signals through the brain. It takes a large number of input features and transform features through layers connected with trainable weights. The weights of these layers are trained by forward propagation of training data then backward propagation of its errors. Back propagation starts from evaluating the difference between prediction and true value, and back calculates the weight adjustment needed to every layer before. We can control the training speed and the complexity of the architecture by tuning the hyper-parameters, such as learning rate and network density.

3.2.4.Output Layer

Output from the last hidden layer is connected to output layer having seven distinct classes. Using SoftMax activation function, output is obtained using the probabilities for each of the seven class. The class with the highest probability is the predicted class.

CHAPTER 4

IMPLEMENTATION

We have used Python Programming Language for implementation purpose. Major Steps are:

i. Data pre-processing
ii. Feature Extraction
iii. Training and Validation

4.1 Data pre-processing

- The fer2013.csv dataset is used which consists of three columns namely emotion, pixels and purpose.
- The column in pixel first of all is stored in a list format.
- Since computational complexity is high for computing pixel values in the range of (0-255), the data in pixel field is normalized to values between [0-1].
- The face objects stored are reshaped and resized to the mentioned size of 48 X 48.
- The respective emotion labels and their respective pixel values are stored in objects.
- Scikit-learn' s train_test_split () function to split the dataset into training and testing data. The test_size is 0.2 i.e., 20% of data is for validation while 80% of the data will be trained.

4.2 Feature Extraction

Haar Cascade classifier is used for feature extraction purpose.

Fig 5: HaarCascade Classifier (Frontal Face)

Cascade function is trained from a lot of images both positive and negative. Based on the training it is then used to detect the objects in the other images. They are huge individual .xml files with a lot of feature sets and each xml corresponds to a very specific type of use case and in our case, we have used frontal-face haar cascade detector to detect front faces.

The algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it.

For this, haar features are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle. Now all possible sizes and locations of each kernel is used to calculate plenty of features. For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. It makes things super-fast.

We apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and

non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features, i.e. a reduction from 160000+ features to 6000 features.

4.3 Training

In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feedforward 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

LeNet is one of the very first convolutional neural networks which helped propel the field of Deep Learning. LeNet Architecture with modification in its pooling part is used to train the model.

Major Operations involved in the training are as follows:

4.3.1 Convolution:

The convolution layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNN might have size 3x5x5 (i.e. images have depth 3 i.e. the colour channels, 5 pixels width and height). During the forward pass, each filter is convolved across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As the filter convolve over the width and height of the input volume it produces a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as a three edge of some orientation or a blotch of some colour on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, there will be an entire set of filters in each convolution layer (e.g. 20 filters), and each of them will produce a separate 2-dimensional activation map.

A filter convolves with the input image to produce a feature map. The convolution of another filter over the same image gives a different feature map. Convolution operation captures the local dependencies in the original image. A CNN learns the values of these filters on its own during the training process (although parameters such as number of filters, filter size, architecture of the network etc. still needed to specify before the training process). The greater number of filters, the more image features get extracted and the better network becomes at recognizing patterns in unseen images.

The size of the Feature Map (Convolved Feature) is controlled by three parameters :

- Depth: Depth corresponds to the number of filters we use for the convolution operation.
- Stride: Stride is the size of the filter, if the size of the filter is 5x5 then stride is 5.
- Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that filter can be applied to bordering elements of input image matrix.

Using zero padding size of the feature map can be controlled

4.3.2 Rectified Linear Unit:

An additional operation called ReLU has been used after every Convolution operation. A Rectified Linear Unit (ReLU) is a cell of a neural network which uses the following activation function to calculate its output given x:

$$R(x) = Max(0,x)$$

When initializing the weights uniformly, half of the weights are negative. This helps creating a sparse feature representation. Another positive aspect is the relatively cheap computation. No exponential function has to be calculated. This function also prevents the vanishing gradient error, since the gradients are linear functions or zero but in no case nonlinear functions.

4.3.3 Maximum Pooling (sub-sampling) :

Spatial Pooling (also called subsampling or down sampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types:

Max, Average, Sum etc. In this case Max Pooling is used in which a spatial neighbourhood (for example, a
2×2 window) is defined and the largest element is taken from the rectified feature map within that window. Max Pooling reduces the input by applying the maximum function over the input xi.

Let m be the size of the filter, then the output calculates as follows:

$$M(xi) = max\ \{xi+k,+l\ |k| \leq m/2\ ,\ |l| \leq m/2\ k,\ l\epsilon\ \mathbb{N}\}$$

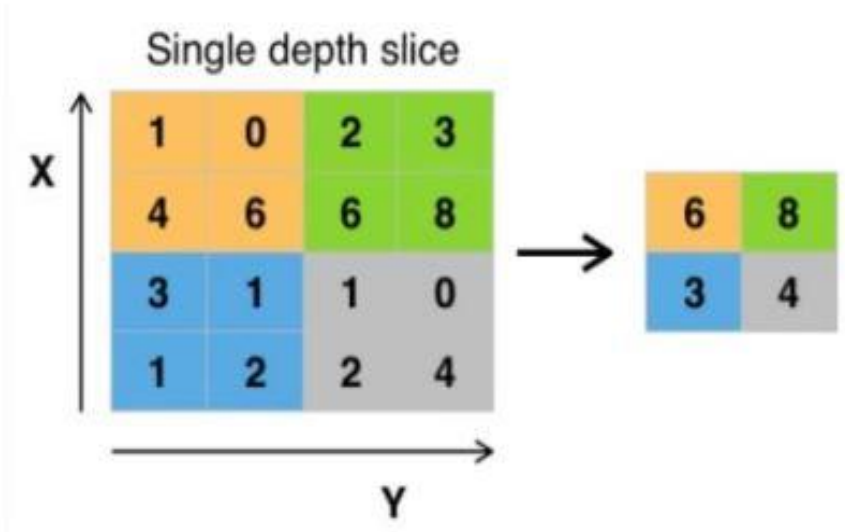

Figure 6 : Max Pooling

In particular, max pooling :

- Makes the input representations (feature dimension) smaller and more manageable.
- Reduces the number of parameters and computations in the network, therefore, controlling over-fitting.
- Makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling.

- Helps us arrive at an almost scale invariant representation. This is very powerful since objects can be detected in an image no matter where they are located.

4.3.4 Classification (Multilayer Perceptron):

The Fully Connected layer is a traditional Multi-Layer Perceptron that uses SoftMax activation function in the output layer. The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

SoftMax is used for activation function. It treats the outputs as scores for each class. In the SoftMax, the function mapping stayed unchanged and these scores are interpreted as the unnormalized log probabilities for each class. SoftMax is calculated as:

$$f(z)_j = \frac{exp(zj)}{\sum_{k=1}^{K} exp(zk)}$$

where j is index for image and K is number of total facial expression class.

Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better. The sum of output

probabilities from the Fully Connected Layer is 1. This is ensured by using

the as the activation function in the output layer of the Fully Connected Layer. The SoftMax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.

CHAPTER 5

RESULT AND ANALYSIS

CNN architecture for facial expression recognition as mentioned above was implemented in Python. Along with Python programming language, NumPy, Pandas and scikit libraries were used.

The testing of the model is carried out using 11960 images. The classifier provided 66.77 % accuracy. The confusion matrix for seven facial expression classes is shown below:

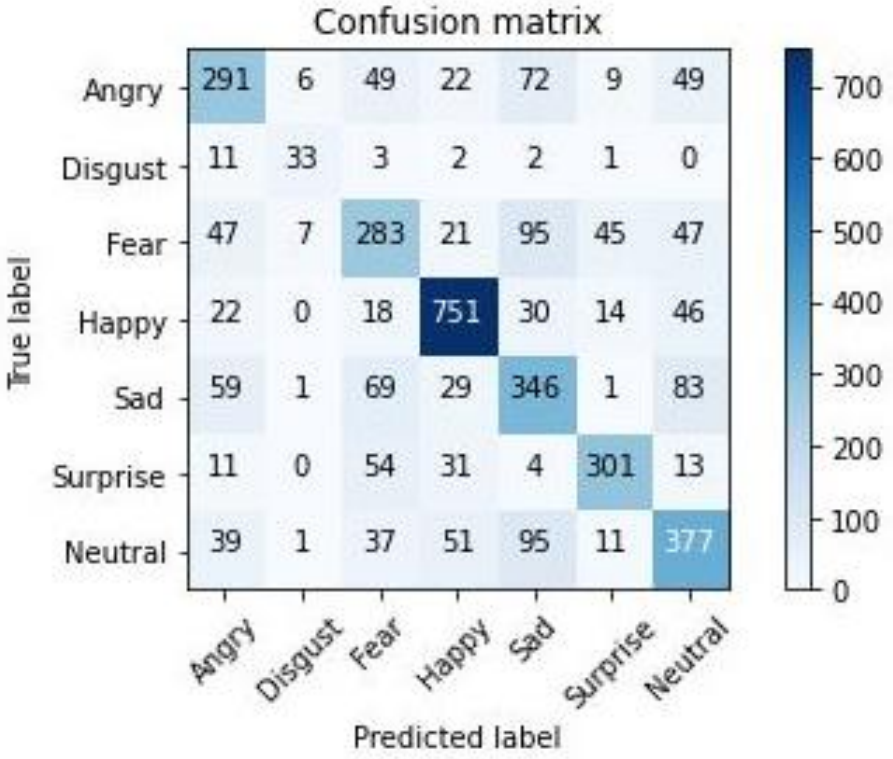

Figure 7: Confusion Matrix

Model performance seems weaker across negative emotions on average. Model frequently misclassified angry, fear and neutral as sad. In addition, it is most confused when predicting sad and neutral faces because these two emotions are probably the least expressive (excluding crying faces).

CNN Classifier will then be used to classify image taken from webcam in Laptop. Face is detected in webcam frames using Haar cascade classifier from OpenCV. Then detected face is cropped and normalized and fed to CNN Classifier.

Usage	Emotion	% in total
Private Test	3589	10.00%
Public Test	3589	10.00%
Training	28709	80.00%
Grand Total	**35887**	**100.00%**

Table 2: Set of pixels v/s usage

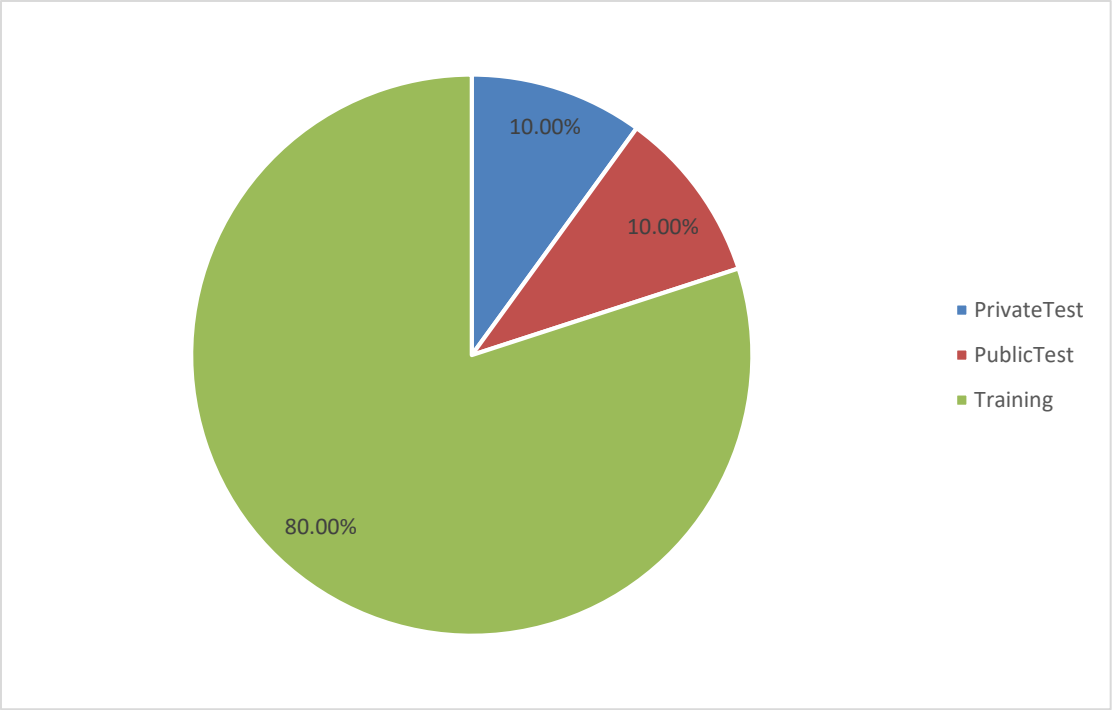

Figure 8: Training, Validation, Test Dataset Distribution

Usage	Emotion	Number of Emotion	% distribution on dataset
Private Test			
	0	491	1.37%
	1	55	0.15%
	2	528	1.47%
	3	879	2.45%
	4	594	1.66%
	5	416	1.16%
	6	626	1.74%
	Total	3589	10.00%
Public Test			
	0	467	1.30%
	1	56	0.16%
	2	496	1.38%
	3	895	2.49%
	4	653	1.82%
	5	415	1.16%
	6	607	1.69%
	Total	3589	10.00%
Training			
	0	3995	11.13%
	1	436	1.21%
	2	4097	11.42%
	3	7215	20.10%
	4	4830	13.46%
	5	3171	8.84%
	6	4965	13.84%
	Total	28709	80.00%
Grand Total		**35887**	**100.00%**

Table 3: Emoticon -wise distribution in dataset

CHAPTER 6

HARDWARE AND SOFTWARE REQUIREMENTS

As the project is developed in python, we have used Anaconda for Python 3.6.5 and Spyder.

- Anaconda

  It is a free and open source distribution of the Python and R programming languages for data science and machine learning related applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution is used by over 6 million users, and it includes more than 250 popular data science packages suitable for Windows, Linux, and MacOS.

- Spyder

  Spyder (formerly Pydee) is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates NumPy, SciPy, Matplotlib and IPython, as well as other open source software. It is released under the MIT license.
  Spyder is extensible with plugins, includes support for interactive tools for data inspection and embeds Python-specific code quality assurance and introspection instruments, such as Pyflakes, Pylint and Rope. It is available cross-platform through Anaconda, on Windows with WinPython and Python (x,y), on macOS through MacPorts, and on major Linux distributions such as Arch Linux, Debian, Fedora, Gentoo Linux, openSUSE and Ubuntu.

Features include:

- editor with syntax highlighting and introspection for code completion
- support for multiple Python consoles (including IPython)
- the ability to explore and edit variables from a GUI

Available plugins include:

- Static Code Analysis with Pylint
- Code Profiling
- Conda Package Manager with Conda

Hardware Interfaces

1. Processor : Intel CORE i5 processor with minimum 2.9 GHz speed.
2. RAM : Minimum 4 GB.
3. Hard Disk : Minimum 500 GB

Software Interfaces

1. Microsoft Word 2003
2. Database Storage : Microsoft Excel
3. Operating System : Windows10

CHAPTER 7

IMAGE INPUTS AND RESULTS

We have taken images as user input in four different ways. The methods and the corresponding results are as follow:

1. Pre-clicked Image

Fig 8: Input- Pre-clicked Image

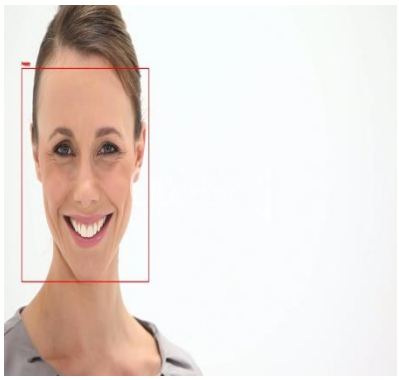

Fig 9 : Output-Prclicked image

Expression correctly identified as happy

2. Pre-recorded Video

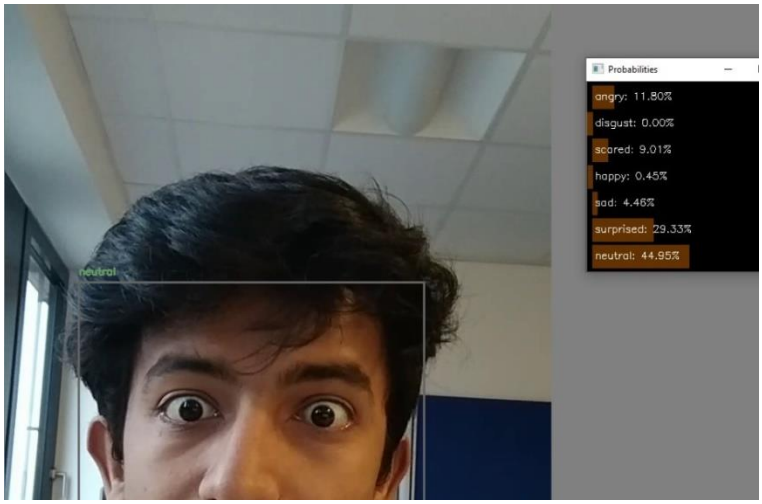

Fig 10: Input1-Pre-recorded video

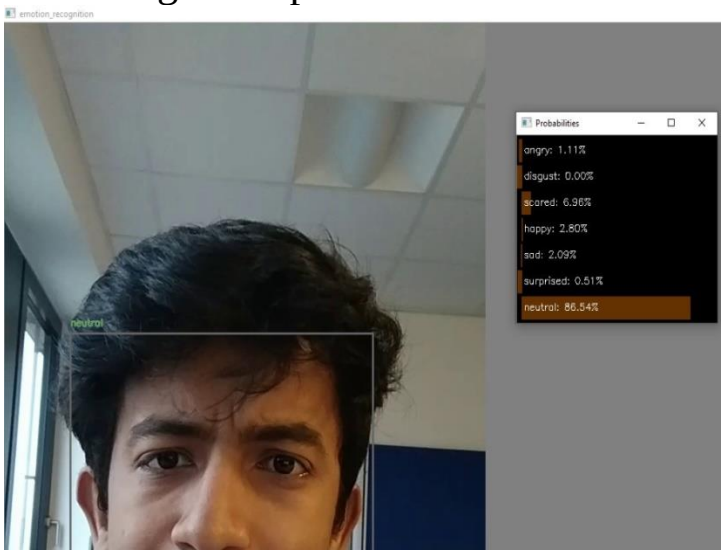

Fig11: Input2-Pre-recorded video

Identification of emotions in a recorded video

4. Image taken live through webcam

Fig 12: Input –Image through webcam

Fig13: Output-Image through webcam

Image gets wrongly identified as angry instead of neutral

4. Live video through webcam

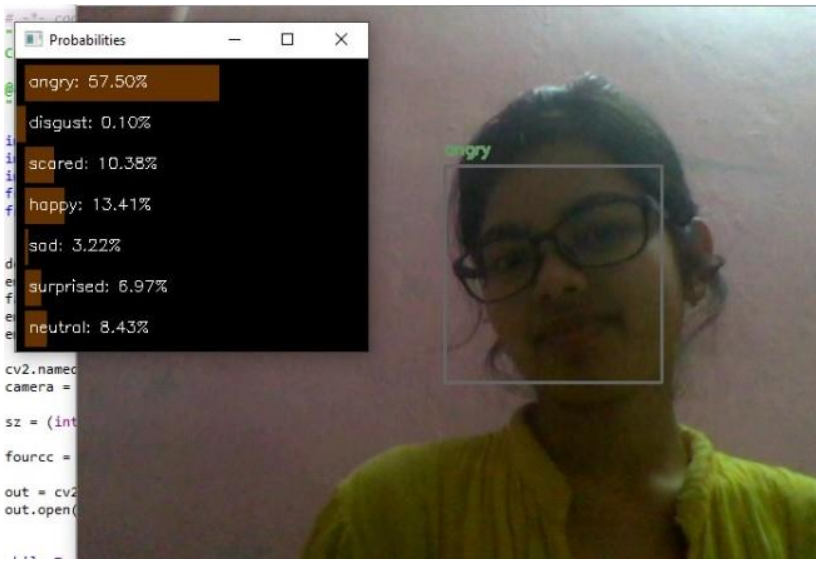

Fig14: Live webcam recorded video

Identification of emotions in a live video

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

8.1 Conclusion of work

In this project, a LeNet architecture based six-layer convolution neural network is implemented to classify human facial expressions i.e. happy, sad, surprise, fear, anger, disgust, and neutral. The classifier achieved accuracy of 66.67%.

8.2 Future scope

In the future work, the model can be extended to colour images. This will allow to investigate the efficacy of pre- trained models such as AlexNet or VGCNet for facial emotion recognition.

REFERENCES

Journal / Conference Papers

[1] Shan, C., Gong, S., & McOwan, P. W. (2005, September). Robust facial expression recognition using local binary patterns. In *Image Processing*, 2005. ICIP 2005. IEEE International Conference on (Vol. 2, pp. II-370). IEEE.

[2] Chibelushi, C. C., & Bourel, F. (2003). Facial expression recognition: A brief tutorial overview. CVonline: On-Line Compendium of Computer Vision, 9

[3] "Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation". DeepLearning 0.1. LISA Lab. Retrieved 31 August

ANNEXURE

Source code of different processes used in the project :

```
"""
Created on Tue Feb 16 09:34:36 2021

@author: S Rani
"""
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")

data = pd.read_csv('./fer2013.csv')

width, height = 48, 48

datapoints = data['pixels'].tolist()

#getting features for training
X = []
for xseq in datapoints:
    xx = [int(xp) for xp in xseq.split(' ')]
    xx = np.asarray(xx).reshape(width, height)
    X.append(xx.astype('float32'))

X = np.asarray(X)
X = np.expand_dims(X, -1)
#getting labels for training
y = pd.get_dummies(data['emotion']).values

#storing them using numpy
np.save('fdataX', X)
np.save('flabels', y)

print("Preprocessing Done")
print("Number of Features: "+str(len(X[0])))
print("Number of Labels: "+ str(len(y[0])))
print("Number of examples in dataset: "+str(len(X)))
print("X,y stored in fdataX.npy and flabels.npy respectively")

```

1.Preprocessing

```
import sys, os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.callbacks import ReduceLROnPlateau, TensorBoard, EarlyStopping, ModelCheckpoint
from keras.models import load_model
from keras.models import model_from_json


num_features = 64
num_labels = 7
batch_size = 64
epochs = 100
width, height = 48, 48

x = np.load('./fdataX.npy')
y = np.load('./flabels.npy')

x -= np.mean(x, axis=0)
x /= np.std(x, axis=0)

#for xx in range(10):
#    plt.figure(xx)
#    plt.imshow(x[xx].reshape((48, 48)), interpolation='none', cmap='gray')
#plt.show()

#splitting into training, validation and testing data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.1, random_state=41)

```

```
36
37 #saving the test samples to be used later
38 np.save('modXtest', X_test)
39 np.save('modytest', y_test)
40
41 #desinging the CNN
42 model = Sequential()
43
44 model.add(Conv2D(num_features, kernel_size=(3, 3), activation='relu', input_shape=(width, height, 1), data_format='channels_last',
45                  kernel_regularizer=l2(0.01)))
46 model.add(Conv2D(num_features, kernel_size=(3, 3), activation='relu', padding='same'))
47 model.add(BatchNormalization())
48 model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
49 model.add(Dropout(0.5))
50
51 model.add(Conv2D(2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
52 model.add(BatchNormalization())
53 model.add(Conv2D(2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
54 model.add(BatchNormalization())
55 model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
56 model.add(Dropout(0.5))
57
58 model.add(Conv2D(2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
59 model.add(BatchNormalization())
60 model.add(Conv2D(2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
61 model.add(BatchNormalization())
62 model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
63 model.add(Dropout(0.5))
64
65 model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
66 model.add(BatchNormalization())
67 model.add(Conv2D(2*2*2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
68 model.add(BatchNormalization())
69 model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
70 model.add(Dropout(0.5))
71
72 model.add(Flatten())
```

```
model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(num_labels, activation='softmax'))

#model.summary()

#Compliling the model with adam optimixer and categorical crossentropy loss
model.compile(loss=categorical_crossentropy,
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7),
              metrics=['accuracy'])

#training the model
model.fit(np.array(X_train), np.array(y_train),
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(np.array(X_valid), np.array(y_valid)),
          shuffle=True)

#saving the  model to be used later
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")

```

2.Training

```
from __future__ import division
from keras.models import model_from_json
import numpy
import numpy as np

json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model_weights.h5")
print("Loaded model from disk")

truey=[]
predy=[]
x = np.load('./modXtest.npy')
y = np.load('./modytest.npy')

yhat= loaded_model.predict(x)
yh = yhat.tolist()
yt = y.tolist()
count = 0

for i in range(len(y)):
    yy = max(yh[i])
    yyt = max(yt[i])
    predy.append(yh[i].index(yy))
    truey.append(yt[i].index(yyt))
    if(yh[i].index(yy)== yt[i].index(yyt)):
        count+=1

acc = (count/len(y))*100

#saving values for confusion matrix and analysis
np.save('truey', truey)
np.save('predy', predy)
print("Predicted and true label values saved")
print("Accuracy on test set :"+str(acc)+"%")
```

3.Testing

```python
import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

y_true = np.load('./truey.npy')
y_pred = np.load('./predy.npy')
cm = confusion_matrix(y_true, y_pred)
labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
title='Confusion matrix'
print(cm)

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
fmt = 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()

```

4.Confusion Matrix

```
7
8 from keras.preprocessing.image import img_to_array
9 from keras.models import load_model
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13
14 detection_model_path = 'haarcascade_frontalface_default.xml'
15 image_path = 'test_for_image.jpg'
16
17 face_detection = cv2.CascadeClassifier(detection_model_path)
18
19 emotion_classifier = load_model("model.hdf5")
20
21 emotions = ['angry', 'disgust', 'scared', 'happy', 'sad', 'surprised', 'neutral']
22
23 color_frame = cv2.imread(image_path)
24 gray_frame = cv2.imread(image_path, 0)
25
26
27 cv2.imshow('Input test image', color_frame)
28 cv2.waitKey(10000)
29 cv2.destroyAllWindows()
30
31
32 detected_faces = face_detection.detectMultiScale(color_frame, scaleFactor=1.1, minNeighbors=5,
33                                                  minSize=(30,30), flags=cv2.CASCADE_SCALE_IMAGE)
34 print('Number of faces detected : ', len(detected_faces))
35
```

```
35
36 if len(detected_faces)>0:
37
38     detected_faces = sorted(detected_faces, reverse=True, key=lambda x: (x[2]-x[0])*(x[3]-x[1]))[0] # if more than one faces
39     (fx, fy, fw, fh) = detected_faces
40
41     im = gray_frame[fy:fy+fh, fx:fx+fw]
42     im = cv2.resize(im, (48,48))  # the model is trained on 48*48 pixel image
43     im = im.astype("float")/255.0
44     im = img_to_array(im)
45     im = np.expand_dims(im, axis=0)
46
47     preds = emotion_classifier.predict(im)[0]
48     emotion_probability = np.max(preds)
49     label = emotions[preds.argmax()]
50
51     cv2.putText(color_frame, label, (fx, fy-10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
52     cv2.rectangle(color_frame, (fx, fy), (fx + fw, fy + fh),(0, 0, 255), 2)
53
54 cv2.imshow('Input test image', color_frame)
55 cv2.imwrite('output_'+image_path.split('/')[-1], color_frame)
56 cv2.waitKey(10000)
57 cv2.destroyAllWindows()
58
59 import matplotlib.image as mpimg
60 img = mpimg.imread('output_for_image.jpg')
61 imgplot = plt.imshow(img)
62 plt.show()
63
64
```

5.Pre-Clicked Image Recognition

```
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import cv2
import numpy as np
import matplotlib.pyplot as plt

videoCaptureObject = cv2.VideoCapture(0)
result=True
while(result):
    ret,frame=videoCaptureObject.read()
    cv2.imwrite("test_for_camera.jpg",frame)
    result=False
videoCaptureObject.release()
cv2.destroyAllWindows()


detection_model_path = 'haarcascade_frontalface_default.xml'
image_path = 'test_for_camera.jpg'

face_detection = cv2.CascadeClassifier(detection_model_path)

emotion_classifier = load_model("model.hdf5")

emotions = ['angry', 'disgust', 'scared', 'happy', 'sad', 'surprised', 'neutral']

color_frame = cv2.imread(image_path)
gray_frame = cv2.imread(image_path, 0)


cv2.imshow('Input test image', color_frame)
cv2.waitKey(10000)
cv2.destroyAllWindows()


detected_faces = face_detection.detectMultiScale(color_frame, scaleFactor=1.1, minNeighbors=5,
                                        minSize=(30,30), flags=cv2.CASCADE_SCALE_IMAGE)
print('Number of faces detected : ', len(detected_faces))

```

```
if len(detected_faces)>0:

    detected_faces = sorted(detected_faces, reverse=True, key=lambda x: (x[2]-x[0])*(x[3]-x[1]))[0] # if more than one faces
    (fx, fy, fw, fh) = detected_faces

    im = gray_frame[fy:fy+fh, fx:fx+fw]
    im = cv2.resize(im, (48,48))  # the model is trained on 48*48 pixel image
    im = im.astype("float")/255.0
    im = img_to_array(im)
    im = np.expand_dims(im, axis=0)

    preds = emotion_classifier.predict(im)[0]
    emotion_probability = np.max(preds)
    label = emotions[preds.argmax()]

    cv2.putText(color_frame, label, (fx, fy-10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
    cv2.rectangle(color_frame, (fx, fy), (fx + fw, fy + fh),(0, 0, 255), 2)

cv2.imshow('Input test image', color_frame)
cv2.imwrite('output_'+image_path.split('/')[-1], color_frame)
cv2.waitKey(10000)
cv2.destroyAllWindows()

import matplotlib.image as mpimg
img = mpimg.imread('output_test_for_camera.jpg')
imgplot = plt.imshow(img)
plt.show()
```

6.Webcam Image Recognition

```
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import imutils
import cv2
import numpy as np


detection_model_path = 'haarcascade_frontalface_default.xml'
emotion_recognition_model_path = 'model.hdf5'
face_detection = cv2.CascadeClassifier(detection_model_path)
emotion_classifier = load_model(emotion_recognition_model_path)
emotions = ['angry', 'disgust', 'scared', 'happy', 'sad', 'surprised', 'neutral']


cv2.namedWindow('emotion_recognition')
camera = cv2.VideoCapture('detect_video_emoticon.mp4')

sz = (int(camera.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(camera.get(cv2.CAP_PROP_FRAME_HEIGHT)))

fourcc = cv2.VideoWriter_fourcc(*'mpeg')

out = cv2.VideoWriter()
out.open('video_emoticon_detected.mp4',fourcc, 15, sz, True) # initialize the writer

while(camera.read()[0]):  # when reading from a video file
    color_frame = camera.read()[1]
    color_frame = imutils.resize(color_frame,width=min(720, color_frame.shape[1]))


    gray_frame = cv2.cvtColor(color_frame, cv2.COLOR_BGR2GRAY)
    detected_faces = face_detection.detectMultiScale(gray_frame,scaleFactor=1.1,minNeighbors=5,minSize=(30,30),flags=cv2.CASCADE_SCALE_IMAGE)

    canvas = np.zeros((250, 300, 3), dtype="uint8")
    frameClone = color_frame.copy()

```

```
    if len(detected_faces)>0:

        detected_faces = sorted(detected_faces, reverse=True, key=lambda x: (x[2]-x[0])*(x[3]-x[1]))[0] #
        (fx, fy, fw, fh) = detected_faces

        im = gray_frame[fy:fy+fh, fx:fx+fw]
        im = cv2.resize(im, (48,48))  # the model is trained on 48*48 pixel image
        im = im.astype("float")/255.0
        im = img_to_array(im)
        im = np.expand_dims(im, axis=0)

        preds = emotion_classifier.predict(im)[0]
        emotion_probability = np.max(preds)
        label = emotions[preds.argmax()]

        cv2.putText(color_frame, label, (fx, fy-10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
        cv2.rectangle(color_frame, (fx, fy), (fx + fw, fy + fh),(0, 0, 255), 2)


    for (i, (emotion, prob)) in enumerate(zip(emotions, preds)):
        # construct the label text
        text = "{}: {:.2f}%".format(emotion, prob * 100)
        w = int(prob * 300)

        cv2.rectangle(canvas, (7, (i * 35) + 5), (w, (i * 35) + 35), (0, 50, 100), -1)
        cv2.putText(canvas, text, (10, (i * 35) + 23), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 255), 1)
        cv2.putText(frameClone, label, (fx, fy - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (100, 150, 100), 2)
        cv2.rectangle(frameClone, (fx, fy), (fx + fw, fy + fh), (100, 100, 100), 2)

    out.write(frameClone)
    out.write(canvas)
    cv2.imshow('emotion_recognition', frameClone)
    cv2.imshow("Probabilities", canvas)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
camera.release()
out.release()
cv2.destroyAllWindows()
```

7. Pre-recorded Video Recognition

```
import cv2
import imutils
import numpy as np
from keras.models import load_model
from keras.preprocessing.image import img_to_array


detection_model_path = 'haarcascade_frontalface_default.xml'
emotion_recognition_model_path = 'model.hdf5'
face_detection = cv2.CascadeClassifier(detection_model_path)
emotion_classifier = load_model(emotion_recognition_model_path)
emotions = ['angry', 'disgust', 'scared', 'happy', 'sad', 'surprised', 'neutral']

cv2.namedWindow('emotion_recognition')
camera = cv2.VideoCapture(0)  ## to use laptop camera

sz = (int(camera.get(cv2.CAP_PROP_FRAME_WIDTH)),int(camera.get(cv2.CAP_PROP_FRAME_HEIGHT)))

fourcc = cv2.VideoWriter_fourcc(*'mpeg')

out = cv2.VideoWriter()
out.open('output_various_emotions.mp4',fourcc, 15, sz, True) # initialize the writer


while True: # when reading from a video camera
    color_frame = camera.read()[1]
    color_frame = imutils.resize(color_frame,width=min(720, color_frame.shape[1]))


    gray_frame = cv2.cvtColor(color_frame, cv2.COLOR_BGR2GRAY)
    detected_faces = face_detection.detectMultiScale(gray_frame,scaleFactor=1.1,minNeighbors=5,minS

    canvas = np.zeros((250, 300, 3), dtype="uint8")
    frameClone = color_frame.copy()

```

```
    if len(detected_faces)>0:

        detected_faces = sorted(detected_faces, reverse=True, key=lambda x: (x[2]-x[0])*(x[3]-x[1]))
        (fx, fy, fw, fh) = detected_faces

        im = gray_frame[fy:fy+fh, fx:fx+fw]
        im = cv2.resize(im, (48,48))  # the model is trained on 48*48 pixel image
        im = im.astype("float")/255.0
        im = img_to_array(im)
        im = np.expand_dims(im, axis=0)

        preds = emotion_classifier.predict(im)[0]
        emotion_probability = np.max(preds)
        label = emotions[preds.argmax()]

        cv2.putText(color_frame, label, (fx, fy-10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2
        cv2.rectangle(color_frame, (fx, fy), (fx + fw, fy + fh),(0, 0, 255), 2)


    for (i, (emotion, prob)) in enumerate(zip(emotions, preds)):
        # construct the label text
        text = "{}: {:.2f}%".format(emotion, prob * 100)
        w = int(prob * 300)

        cv2.rectangle(canvas, (7, (i * 35) + 5), (w, (i * 35) + 35), (0, 50, 100), -1)
        cv2.putText(canvas, text, (10, (i * 35) + 23), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 2
        cv2.putText(frameClone, label, (fx, fy - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (100, 150, 10
        cv2.rectangle(frameClone, (fx, fy), (fx + fw, fy + fh), (100, 100, 100), 2)

    out.write(frameClone)
    out.write(canvas)
    cv2.imshow('emotion_recognition', frameClone)
    cv2.imshow("Probabilities", canvas)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
camera.release()
out.release()
cv2.destroyAllWindows()
```

8. Live Webcam Recorded Video Recognition