**SAMPLE CODE**

**Training\_images.py**

import cv2, sys, numpy, os, time

count = 0

size = 4

fn\_haar = 'haarcascade\_frontalface\_default.xml'

fn\_dir = 'database'

fn\_name = sys.argv[1] # name of the person

path = os.path.join(fn\_dir, fn\_name)

if not os.path.isdir(path):

os.mkdir(path)

(im\_width, im\_height) = (112, 92)

haar\_cascade = cv2.CascadeClassifier(fn\_haar)

webcam = cv2.VideoCapture(0)

print("--------------------Training Process Started------------------")

print("--------------------Give some expressions---------------------")

while count < 45:

(rval, im) = webcam.read()

im = cv2.flip(im, 1, 0)

gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

mini = cv2.resize(gray, (int(gray.shape[1] / size), int(gray.shape[0] / size)))

faces = haar\_cascade.detectMultiScale(mini)

faces = sorted(faces, key=lambda x: x[3])

if faces:

face\_i = faces[0]

(x, y, w, h) = [v \* size for v in face\_i]

face = gray[y:y + h, x:x + w]

face\_resize = cv2.resize(face, (im\_width, im\_height))

pin = sorted([int(n[:n.find('.')]) for n in os.listdir(path)

if n[0] != '.'] + [0])[-1] + 1

cv2.imwrite('%s/%s.png' % (path, pin), face\_resize)

cv2.rectangle(im, (x, y), (x + w, y + h), (0, 255, 0), 3)

cv2.putText(im, fn\_name, (x - 10, y - 10), cv2.FONT\_HERSHEY\_PLAIN,

1, (0, 255, 0))

time.sleep(0.38)

count += 1

cv2.imshow('OpenCV', im)

key = cv2.waitKey(10)

if key == 27:

break

print(str(count) + " images taken and saved to " + fn\_name + " folder in database ")

**face\_recognize.py**

import cv2, sys, numpy, os

import time

size = 4

capture\_duration = 3

haar\_file = 'haarcascade\_frontalface\_default.xml'

datasets = 'database'

# Part 1: Create fisherRecognizer

print('Training...')

# Create a list of images and a list of corresponding names

(images, lables, names, id) = ([], [], {}, 0)

for (subdirs, dirs, files) in os.walk(datasets):

for subdir in dirs:

names[id] = subdir

subjectpath = os.path.join(datasets, subdir)

for filename in os.listdir(subjectpath):

path = subjectpath + '\\' + filename

#print("Path = ",path)

lable = id

images.append(cv2.imread(path, 0))

lables.append(int(lable))

id += 1

(width, height) = (130, 100)

# Create a Numpy array from the two lists above

(images, lables) = [numpy.array(lis) for lis in [images, lables]]

# OpenCV trains a model from the images

# NOTE FOR OpenCV2: remove '.face'

#model = cv2.createFisherFaceRecognizer()

def faceDetectStatus():

try:

# model = cv2.face.FisherFaceRecognizer\_create()

# model = cv2.face.LBPHFaceRecognizer\_create()

# create our LBPH face recognizer

model = cv2.face.LBPHFaceRecognizer\_create()

# or use EigenFaceRecognizer by replacing above line with

# model = cv2.face.EigenFaceRecognizer\_create()

# or use FisherFaceRecognizer by replacing above line with

# model = cv2.face.FisherFaceRecognizer\_create()

model.train(images, lables)

except Exception as e:

print(str(e))

pass

# Part 2: Use fisherRecognizer on camera stream

face\_cascade = cv2.CascadeClassifier(haar\_file)

webcam = cv2.VideoCapture(0)

# save current time

prev\_time = time.time()

# start webcam feed

start\_time = time.time()

# while True:

while (int(time.time() - start\_time) < capture\_duration):

(\_, im) = webcam.read()

gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

rslt = []

for (x, y, w, h) in faces:

cv2.rectangle(im, (x, y), (x + w, y + h), (255, 0, 0), 2)

face = gray[y:y + h, x:x + w]

face\_resize = cv2.resize(face, (width, height))

# Try to recognize the face

prediction = model.predict(face\_resize)

print('Predection is ', prediction)

cv2.rectangle(im, (x, y), (x + w, y + h), (0, 255, 0), 3)

if prediction[1] > 90:

rslt.append('True')

cv2.putText(im, '%s - %.0f' % (names[prediction[0]], prediction[1]), (x - 10, y - 10),

cv2.FONT\_HERSHEY\_PLAIN, 1, (0, 255, 0))

break

else:

rslt.append('False')

cv2.putText(im, 'not recognized', (x - 10, y - 10), cv2.FONT\_HERSHEY\_PLAIN, 1, (0, 255, 0))

break

cv2.imshow('OpenCV', im)

key = cv2.waitKey(10)

if key == 27:

# if key == -1:

break

cv2.destroyAllWindows()

return rslt

**eyedetection.py**

"""

This module is a face detection classifier and turns on the webcam.

"""

import os

import time

import cv2

# Get the path of the this script

CURRENT\_FILE\_PATH = os.path.dirname(\_\_file\_\_)

# Load the haar-like features

FACE\_CASCADE = cv2.CascadeClassifier(os.path.join(

CURRENT\_FILE\_PATH, 'haarcascade\_frontalface\_default.xml'))

EYE\_CASCADE = cv2.CascadeClassifier(

os.path.join(CURRENT\_FILE\_PATH, 'haarcascade\_eye.xml'))

SMILE\_CASCADE = cv2.CascadeClassifier(

os.path.join(CURRENT\_FILE\_PATH, 'haarcascade\_smile.xml'))

def face\_detection(bw\_img, orig\_img):

"""Takes the black-white version of an image and the original image and

performs face, eye and smile detection on the black-white image. The detected features

are drawn with rectangles on the original image.

:param bw\_img: black and white image from original image

:param orig\_img: original image

:type bw\_img: <class 'numpy.ndarray'>

:type orig\_img: <class 'numpy.ndarray'>

:return: returns orig\_img with rectangles on regions of interest

:rtype: <class 'numpy.ndarray'>

"""

faces = FACE\_CASCADE.detectMultiScale(bw\_img, 1.3, 5)

for fx, fy, fw, fh in faces:

cv2.rectangle(orig\_img, (fx, fy), (fx+fw, fy+fh), (255, 0, 0), 2)

region\_of\_interest\_bw = bw\_img[fy:fy+fh, fx:fx+fw]

region\_of\_interest\_color = orig\_img[fy:fy+fh, fx:fx+fw]

eyes = EYE\_CASCADE.detectMultiScale(region\_of\_interest\_bw, 1.1, 22)

for ex, ey, ew, eh in eyes:

cv2.rectangle(region\_of\_interest\_color, (ex, ey),

(ex+ew, ey+eh), (0, 255, 0), 2)

smiles = SMILE\_CASCADE.detectMultiScale(region\_of\_interest\_bw, 1.7, 22)

for sx, sy, sw, sh in smiles:

cv2.rectangle(region\_of\_interest\_color, (sx, sy),

(sx+sw, sy+sh), (0, 0, 255), 2)

return orig\_img

def make\_screenshot(img, counter):

"""Takes a screenshot and saves the frame/image

:param img: colored image of the video capture process

:param counter: increased counter so the previously saved images aren't overwritten

:type img: <class 'numpy.ndarray'>

:type counter: int

"""

img\_path = os.path.join(CURRENT\_FILE\_PATH, 'smiles')

if not os.path.exists(img\_path):

os.makedirs(img\_path)

cv2.imwrite(os.path.join(

img\_path, 'screenshot-{0}.jpeg'.format(counter)), img)

def start\_video\_capturing(video\_capture):

"""Starts the webcam and captures a video stream.

Press 's' to take a screenshot of the video stream.

Press 'Esc' to close and exit the video stream.

:param video\_capture: video stream object of the webcam

:type video\_capture: <class 'cv2.VideoCapture'>

"""

capture\_duration = 30

start\_time = time.time()

screenshot\_counter = 0

#while True:

while ( int(time.time() - start\_time) < capture\_duration ):

\_, img = video\_capture.read()

bw\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

canvas = face\_detection(bw\_img, img)

cv2.imshow('Video', canvas)

k = cv2.waitKey(1)

if k == ord('s'):

make\_screenshot(img, screenshot\_counter)

screenshot\_counter += 1

elif k == 27:

break

video\_capture.release()

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

# 0 = internal webcam, 1 = external webcam

VIDEO\_CAPTURE = cv2.VideoCapture(0)

#VIDEO\_CAPTURE = cv2.VideoCapture('http://192.168.1.4:8080/video') '''from Ip Web Cam from Android Device'''

start\_video\_capturing(VIDEO\_CAPTURE)

**webcam\_utils.py**

# utility file for emotion recognition from realtime webcam feed

import cv2

import sys

from keras.models import load\_model

import time

import numpy as np

from decimal import Decimal

from model\_utils import define\_model, model\_weights

from collections import Counter

import speech\_recognition as sr

import pyttsx3

import re

import webbrowser

# loads and resizes an image

def resize\_img(image\_path):

img = cv2.imread(image\_path, 1)

img = cv2.resize(img, (48, 48))

return True

# runs the realtime emotion detection

def realtime\_emotions():

# load keras model

model = define\_model()

model = model\_weights(model)

print('Model loaded')

# save location for image

save\_loc = 'save\_loc/1.jpg'

# numpy matrix for stroing prediction

result = np.array((1,7))

# for knowing whether prediction has started or not

once = False

# load haar cascade for face

faceCascade = cv2.CascadeClassifier(r'haarcascades/haarcascade\_frontalface\_default.xml')

# list of given emotions

EMOTIONS = ['Angry', 'Disgusted', 'Fearful', 'Happy', 'Sad', 'Surprised', 'Neutral']

# store the emoji coreesponding to different emotions

emoji\_faces = []

for index, emotion in enumerate(EMOTIONS):

emoji\_faces.append(cv2.imread('emojis/' + emotion.lower() + '.png', -1))

# set video capture device , webcam in this case

capture\_duration = 20

video\_capture = cv2.VideoCapture(0)

video\_capture.set(3, 640) # WIDTH

video\_capture.set(4, 480) # HEIGHT

# save current time

prev\_time = time.time()

# start webcam feed

start\_time = time.time()

lt = []

#while True: #infinite loop

while( int(time.time() - start\_time) < capture\_duration ):

# Capture frame-by-frame

ret, frame = video\_capture.read()

# mirror the frame

frame = cv2.flip(frame, 1, 0)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# find face in the frame

faces = faceCascade.detectMultiScale(

gray,

scaleFactor=1.1,

minNeighbors=5,

minSize=(30, 30),

flags=cv2.CASCADE\_SCALE\_IMAGE

)

# Draw a rectangle around the faces

for (x, y, w, h) in faces:

# required region for the face

roi\_color = frame[y-90:y+h+70, x-50:x+w+50]

# save the detected face

cv2.imwrite(save\_loc, roi\_color)

# draw a rectangle bounding the face

cv2.rectangle(frame, (x-10, y-70),

(x+w+20, y+h+40), (15, 175, 61), 4)

# keeps track of waiting time for emotion recognition

curr\_time = time.time()

# do prediction only when the required elapsed time has passed

if curr\_time - prev\_time >=1:

# read the saved image

img = cv2.imread(save\_loc, 0)

if img is not None:

# indicates that prediction has been done atleast once

once = True

# resize image for the model

img = cv2.resize(img, (48, 48))

img = np.reshape(img, (1, 48, 48, 1))

# do prediction

result = model.predict(img)

print(EMOTIONS[np.argmax(result[0])])

lt.append(EMOTIONS[np.argmax(result[0])])

#print(lt)

#c = Counter(lt)

#m = c.most\_common(1)

#print(c)

#print("most common",m)

#save the time when the last face recognition task was done

prev\_time = time.time()

if once == True:

total\_sum = np.sum(result[0])

# select the emoji face with highest confidence

emoji\_face = emoji\_faces[np.argmax(result[0])]

for index, emotion in enumerate(EMOTIONS):

text = str(

round(Decimal(result[0][index]/total\_sum\*100), 2) ) + "%"

# for drawing progress bar

cv2.rectangle(frame, (100, index \* 20 + 10), (100 +int(result[0][index] \* 100), (index + 1) \* 20 + 4),

(255, 0, 0), -1)

# for putting emotion labels

cv2.putText(frame, emotion, (10, index \* 20 + 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (7, 109, 16), 2)

# for putting percentage confidence

cv2.putText(frame, text, (105 + int(result[0][index] \* 100), index \* 20 + 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 1)

# overlay emoji on the frame for all the channels

for c in range(0, 3):

# for doing overlay we need to assign weights to both foreground and background

foreground = emoji\_face[:, :, c] \* (emoji\_face[:, :, 3] / 255.0)

background = frame[350:470, 10:130, c] \* (1.0 - emoji\_face[:, :, 3] / 255.0)

frame[350:470, 10:130, c] = foreground + background

#return lt

break

# Display the resulting frame

cv2.imshow('Video', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# When everything is done, release the capture

video\_capture.release()

cv2.destroyAllWindows()

#print("This Is LT ",lt)

return lt

**preprocess.py**

import os

import csv

import argparse

import numpy as np

import scipy.misc

import pandas as pd

import numpy as np

parser = argparse.ArgumentParser()

parser.add\_argument('-f', '--file', required=True, help="path of the csv file")

parser.add\_argument('-o', '--output', required=True, help="path of the output directory")

args = parser.parse\_args()

w, h = 48, 48

image = np.zeros((h, w), dtype=np.uint8)

id = 1

emo\_list = []

with open(args.file, 'r') as csvfile:

datareader = csv.reader(csvfile, delimiter =',')

headers = next(datareader)

print(headers)

for row in datareader:

emotion = row[0]

pixels = map(int, row[1].split())

usage = row[2]

pixels\_array = np.asarray(list(pixels))

image = np.reshape(pixels\_array, (w, h))

stacked\_image = np.dstack((image,) \* 3)

# add emotion to the list

emo\_list += emotion

image\_folder = os.path.join(args.output, usage)

if not os.path.exists(image\_folder):

os.makedirs(image\_folder)

image\_file = os.path.join(image\_folder , str(id) + '.jpg')

scipy.misc.imsave(image\_file, stacked\_image)

id += 1

if id % 100 == 0:

print('Processed {} images'.format(id))

np.savetxt("emotions.csv", emo\_list, delimiter=",", fmt='%s')

print("Finished processing {} images".format(id))

**main.py**

import argparse

from webcam\_utils import realtime\_emotions

from prediction\_utils import prediction\_path

from collections import Counter

from face\_recognize import faceDetectStatus

from smileseyedetect import start\_video\_capturing

import cv2

from RecordConversation import recordUserVoices

from PlayConversation import playUserRecordedAudio

import speech\_recognition as sr

import pyttsx3

import re

import webbrowser

# for running realtime emotion detection

def run\_realtime\_emotion():

lt = realtime\_emotions()

return lt

# to run emotion detection on image saved on disk

def run\_detection\_path(path):

prediction\_path(path)

def main():

rslt = faceDetectStatus()

print('Result iS ',rslt)

if len(rslt) == 0:

print("Authentication Failed")

exit(0)

else:

if rslt[0] == 'True':

print('User Authenitcated Success')

VIDEO\_CAPTURE = cv2.VideoCapture(0)

start\_video\_capturing(VIDEO\_CAPTURE)

else:

print("User Authentication Failed")

exit(0)

parser = argparse.ArgumentParser()

parser.add\_argument("func\_name", type=str,

help="Select a function to run. <emo\_realtime> or <emo\_path>")

parser.add\_argument("--path", default="saved\_images/1.jpg", type=str,

help="Specify the complete path where the image is saved.")

# parse the args

args = parser.parse\_args()

#print('\*\*\*\*ARGS: ' + str(args))

if args.func\_name == "emo\_realtime":

lt = run\_realtime\_emotion()

return lt

elif args.func\_name == "emo\_path":

lt = run\_detection\_path(args.path)

return lt

else:

print("Usage: python main.py <function name>")

'''

#This code will give you the list of microphoine devices

import speech\_recognition as sr

for index, name in enumerate(sr.Microphone.list\_microphone\_names()):

print("Microphone with name \"{1}\" found for `Microphone(device\_index={0})`".format(index, name))

'''

if \_\_name\_\_ == '\_\_main\_\_':

lt = main()

engine = pyttsx3.init()

engine.say("Conversation Recording Stared.. please Make sailence for audio recording")

engine.runAndWait()

recordUserVoices()

playUserRecordedAudio()

engine.say("Thanking You for bare with us.....")

engine.runAndWait()

'''

r = sr.Recognizer()

with sr.Microphone(device\_index = 0) as source:

print("Say Something")

audio = r.listen(source)

print("TIME OVER, THANX ")

try:

x = r.recognize\_google(audio)

engine = pyttsx3.init()

print("TEXT:", x)

engine.say(x)

engine.runAndWait()

term = 'yes'

words = x.split()

if term in words:

print("Python Project")

engine.say('This is ML Project')

engine.runAndWait()

webbrowser.open("notepad.exe", "testurl.py")

#webbrowser.open\_new("https://www.youtube.com/watch?v=DeB5N\_bH7E8")

elif x == 'no':

print("This is ML Project")

else:

print("command not found")

except:

pass;'''

print("This is LT Object ",lt)

c = Counter(lt)

m = c.most\_common(7)

print("most common",m)

for x in m:

#print('Katti Expressions ',x[0])

if x[0]=='Happy':

print("Your feeling of great happiness and pleasure that lifts up the spirit")

engine.say('Your feeling of great happiness and pleasure that lifts up the spirit')

elif x[0]=='Sad':

print("Your may be regret,disappointment for something...!")

engine.say('Your may be regret,disappointment for something...!')

elif x[0] == 'Neutral':

print("It may be caused by a lack of emotion, depression, boredom or slight confusion, such as when someone refers to something which the listener does not understand")

engine.say('It may be caused by a lack of emotion, depression, boredom or slight confusion, such as when someone refers to something which the listener does not understand')

elif x[0] == 'Angry':

print("your worrying about somthing (or) mad about something and not going to take it anymore")

engine.say('your worrying about somthing (or) mad about something and not going to take it anymore')

elif x[0] == 'Disgusted':

print("you’re struggling with a situation, but carrying on through your frustration.")

engine.say('you’re struggling with a situation, but carrying on through your frustration.')

elif x[0] == 'Fearful':

print("your thinking about a thing or person that can harm you. ")

engine.say('your thinking about a thing or person that can harm you. ')

else:

print("your somthing unexcepted shock")

engine.say('your somthing unexcepted shock')

engine.runAndWait()

**speechrecord.py**

# import libraries

import cv2

import SpeecRecogn

input\_movie = cv2.VideoCapture("katti.mp4")

length = int(input\_movie.get(cv2.CAP\_PROP\_FRAME\_COUNT))

image = SpeecRecogn.load\_image\_file("ram.jpg")

face\_encoding = SpeecRecogn.face\_encodings(image)[0]

codec = int(input\_movie.get(cv2.CAP\_PROP\_FOURCC))

fps = int(input\_movie.get(cv2.CAP\_PROP\_FPS))

frame\_width = int(input\_movie.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(input\_movie.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

output\_movie = cv2.VideoWriter("output.mp4", codec, fps, (frame\_width,frame\_height))

known\_faces = [

face\_encoding,

]

# Initialize variables

face\_locations = []

face\_encodings = []

face\_names = []

frame\_number = 0

while True:

# Grab a single frame of video

ret, frame = input\_movie.read()

frame\_number += 1

# Quit when the input video file ends

if not ret:

break

# Convert the image from BGR color (which OpenCV uses) to RGB color (which face\_recognition uses)

rgb\_frame = frame[:, :, ::-1]

# Find all the faces and face encodings in the current frame of video

face\_locations = SpeecRecogn.face\_locations(rgb\_frame, model="cnn")

face\_encodings = SpeecRecogn.face\_encodings(rgb\_frame, face\_locations)

face\_names = []

for face\_encoding in face\_encodings:

# See if the face is a match for the known face(s)

match = SpeecRecogn.compare\_faces(known\_faces, face\_encoding, tolerance=0.50)

name = None

if match[0]:

name = "Phani Srikant"

face\_names.append(name)

# Label the results

for (top, right, bottom, left), name in zip(face\_locations, face\_names):

if not name:

continue

# Draw a box around the face

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# Draw a label with a name below the face

cv2.rectangle(frame, (left, bottom - 25), (right, bottom), (0, 0, 255), cv2.FILLED)

font = cv2.FONT\_HERSHEY\_DUPLEX

cv2.putText(frame, name, (left + 6, bottom - 6), font, 0.5, (255, 255, 255), 1)

# Write the resulting image to the output video file

print("Writing frame {} / {}".format(frame\_number, length))

output\_movie.write(frame)

# All done!

input\_movie.release()

cv2.destroyAllWindows()

**playconversation.py**

import pyaudio

import wave

def playUserRecordedAudio():

filename = 'output.wav'

# Set chunk size of 1024 samples per data frame

chunk = 1024

# Open the sound file

wf = wave.open(filename, 'rb')

# Create an interface to PortAudio

p = pyaudio.PyAudio()

# Open a .Stream object to write the WAV file to

# 'output = True' indicates that the sound will be played rather than recorded

stream = p.open(format=p.get\_format\_from\_width(wf.getsampwidth()), channels=wf.getnchannels(),

rate=wf.getframerate(), output=True)

# Read data in chunks

data = wf.readframes(chunk)

# Play the sound by writing the audio data to the stream

x = 0

# while data != '':

print(type(data))

while data != '':

stream.write(data)

data = wf.readframes(chunk)

x = x + 1

#print(x)

if x == 2000:

break

# Close and terminate the stream

# cls

# exit(0)

stream.close()

p.terminate()