**AUDIO MODULE**

1. **from** **keras.models** **import** load\_model
2. **from** **librosa.feature** **import** melspectrogram
3. num\_to\_labels\_emotion\_audio=np.load('num\_to\_labels\_emotion\_audio.npy').item()
4. audio\_emotion\_model=load\_model('model\_checkpoint\_combinedfeatures\_ravdess\_savdee54\_60.h5')
5. audio\_emotion\_model.summary()
6. **def** get\_emotion\_audio(audio,sr):
7. print(audio)
8. print(sr)
9. S = melspectrogram(audio.astype('float32'), sr=sr)
10. S = 10 \* np.log(S + 1e-15)
11. S=np.expand\_dims(S,axis=0)
13. S=np.expand\_dims(S,axis=3)
14. print(S.shape)
15. emotion=num\_to\_labels\_emotion\_audio[np.argmax(audio\_emotion\_model.predict(S))]
16. print(emotion)
17. **return** emotion
18. num\_to\_labels\_classify\_audio=np.load('num\_to\_labels\_audio\_classify.npy').item()
19. audio\_classify\_model=load\_model('model\_checkpoint\_combinedfeatures\_ravdess\_savdee54\_60.h5')
20. *#audio\_classify\_model.summary()*
21. **def** get\_audio\_classification(audio,sr):
22. print(audio)
23. print(sr)
24. S = melspectrogram(audio.astype('float32'), sr=sr)
25. S = 10 \* np.log(S + 1e-15)
26. S=np.expand\_dims(S,axis=0)
27. S=np.expand\_dims(S,axis=3)
28. *#print(S.shape)*
29. predicted\_prob=audio\_classify\_model.predict(S)
30. audio\_classes=num\_to\_labels\_classify\_audio[np.argmax(predicted\_prob)]
31. print(predicted\_prob)
32. **return** audio\_classes
33. **import** **pyaudio**
34. **import** **numpy** **as** **np**
35. **import** **sounddevice** **as** **sd**
36. **import** **librosa**
37. CHUNK = 48000 *# number of data points to read at a time*
38. RATE = 16000 *# time resolution of the recording device (Hz)*
39. p=pyaudio.PyAudio() *# start the PyAudio class*
40. stream=p.open(format=pyaudio.paInt16,channels=1,rate=RATE,input=**True**,
41. frames\_per\_buffer=CHUNK) *#uses default input device*
42. *# create a numpy array holding a single read of audio data*
43. stop=0
44. suspicious\_sounds=['vacuum\_cleaner','door\_wood\_knock','fireworks','chainsaw','crackling\_fire','engine','hand\_saw','glass\_breaking','siren']
46. human\_presence\_classes=['clapping','mouse\_click','laughing','keyboard\_typing','footsteps','brushing\_teeth','drinking\_sipping','breathing','crying\_baby','coughing','snoring','sneezing']
48. **while** **not** stop: *#to it a few times just to see*
49. *#print('Recording')*
50. audio=np.fromstring(stream.read(CHUNK),dtype=np.int16)
52. *#sd.play(audio,RATE)*
53. *#S = librosa.feature.melspectrogram(audio.astype('float32'), sr=RATE)*
54. *#S = 10 \* np.log(S + 1e-15)*
55. audio\_classes=get\_audio\_classification(audio,RATE)
56. print("[DETECTED AUDIO CLASS] ",audio\_classes)

59. **if** audio\_classes **in** human\_presence\_classes:
60. print('Human Presense detected')
61. em=get\_emotion\_audio(audio,RATE)
62. print("[DETECTED EMOTION] ",em)
63. **elif** audio\_classes **in** suspicious\_sounds:
64. print('[ALERT] Suspicious sound....',audio\_classes)
65. **else**:
66. print("Empty room")

69. stop=1
70. *# close the stream gracefully*
71. stream.stop\_stream()
72. stream.close()
73. p.terminate()

**VIDEO MODULE**

1. # [Navin\_Kumar\_Manaswi]\_Deep\_Learning\_with\_Applicati(z-lib.org).pdf
2. # Multi face tracking : https://www.guidodiepen.nl/2017/02/tracking-multiple-faces/
3. #import gc
4. import cv2
5. from dlib import get\_frontal\_face\_detector, correlation\_tracker, rectangle,shape\_predictor
6. from os import path, getcwd, listdir, walk
7. import numpy as np
8. from face\_recognition import face\_encodings,compare\_faces,load\_image\_file
9. from imageai.Detection import ObjectDetection
10. #from warnings import filterwarnings
11. import datetime
12. import time
13. import imutils
14. from keras.models import load\_model,Model
15. #import dlib
16. from imutils.face\_utils import FaceAligner
17. from imutils.face\_utils import rect\_to\_bb
18. from pickle import load,dump,loads
19. #from numpy import argmax
20. from keras.preprocessing.sequence import pad\_sequences
21. from keras.applications import inception\_resnet\_v2
22. from keras.preprocessing.image import load\_img
23. from keras.preprocessing.image import img\_to\_array
24. #from keras.applications.vgg16 import preprocess\_input
25. from sklearn.feature\_extraction.text import CountVectorizer
26. from sklearn.ensemble import RandomForestClassifier
27. from KaggleWord2VecUtility import KaggleWord2VecUtility
28. import re
29. #filterwarnings("ignore")
30. #from FaceRecogEncodings\_SVM import recognise\_img\_SVM
31. #saved\_faces = []
32. face\_detector = get\_frontal\_face\_detector()
33. object\_detector = ObjectDetection()
34. object\_detector.setModelTypeAsRetinaNet()
35. object\_detector.setModelPath('resnet50\_coco\_best\_v2.1.0.h5')
36. object\_detector.loadModel(detection\_speed='fast')
37. min\_area = 1000
38. #num\_to\_labels = np.load('num\_to\_labels.npy').item()
39. emotion\_model = load\_model('image\_emotion\_fer\_jaffe\_raf\_encodings\_rus\_47.h5')
40. ultimate\_label\_mapping = {
41. 'anger': 0,
42. 'disgust': 1,
43. 'fear': 2,
44. 'happy': 3,
45. 'neutral': 4,
46. 'sad': 5,
47. 'surprise': 6
49. }
50. num\_to\_labels={v:k for k,v in ultimate\_label\_mapping.items()}
51. shape\_predictor\_ = 'shape\_predictor\_68\_face\_landmarks.dat'
52. #detector = dlib.get\_frontal\_face\_detector()
53. predictor = shape\_predictor(shape\_predictor\_)
54. fa = FaceAligner(predictor, desiredFaceWidth=256)
55. def align\_face(image):
56. image = imutils.resize(image, width=200)
57. image=image.astype(np.uint8)
58. if image.ndim >2:
59. gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)
60. else:
61. gray=image
62. rects = face\_detector(gray, 2)
63. # loop over the face detections
64. '''for rect in rects:
65. # extract the ROI of the \*original\* face, then align the face
66. # using facial landmarks
67. (x, y, w, h) = rect\_to\_bb(rect)
68. faceOrig = imutils.resize(image[y:y + h, x:x + w], width=256)
69. faceAligned = fa.align(image, gray, rect)'''
70. if len(rects)>0:
71. faceAligned = fa.align(image, gray, rects[0])
72. else:
73. faceAligned=image
75. return faceAligned
76. def detect\_emotion(image\_):
77. # image=cv2.imread(image)
78. # image=cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)
79. # print(image.shape)
80. #image = cv2.cvtColor(image\_.copy(), cv2.COLOR\_BGR2GRAY)
81. #print(image.shape)
82. image=cv2.resize(image\_.copy(),(256,256))
83. image=align\_face(image.reshape(256,256).astype(np.float32))
84. encoded\_image=face\_encodings(image[:,:,np.newaxis].repeat(3, 2),known\_face\_locations=[(0,image.shape[1],image.shape[0],0)])
85. encoded\_image=np.array(encoded\_image).ravel().reshape((1,128))
86. emotion=num\_to\_labels[int(np.argmax(emotion\_model.predict(encoded\_image)))]
87. '''
88. image = np.expand\_dims(image, axis=2)
89. image = np.expand\_dims(image, axis=0)
90. print(image.shape)
91. y\_pred = np.argmax(emotion\_model.predict(image))
92. print(y\_pred)
93. emotion = num\_to\_labels[y\_pred]'''
94. return emotion
95. def detect\_motion(ref\_img, img):
96. firstFrame = ref\_img
97. img = cv2.resize(img, (500, 500))
98. # firstFrame=imutils.resize(firstFrame,width=500)
99. if img.ndim>2:
100. gray\_frame = cv2.cvtColor(img.copy(), cv2.COLOR\_BGR2GRAY)
101. else:
102. gray\_frame=img
103. gray\_frame = cv2.GaussianBlur(gray\_frame, (21, 21), 0)
104. occupied = 0
105. frameDelta = cv2.absdiff(firstFrame, gray\_frame)
106. #print("FrameDelta shape:",frameDelta.shape)
107. thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH\_BINARY)[1]
108. #print("Threshold shape before diluting",thresh.shape)
109. # Dilate the threshold image to fill in the holes,then find countours on the image
110. thresh = cv2.dilate(thresh, None, iterations=2)
111. #print("Threshold shape After diluting",thresh.shape)
112. #thresh = cv2.cvtColor(thresh, cv2.CV\_8UC1)
113. # print(thresh)
114. # print(thresh.shape)
115. cnts = cv2.findContours(
116. thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)
117. cnts = imutils.grab\_contours(cnts)
118. for c in cnts:
119. if cv2.contourArea(c) < min\_area:
120. continue
121. (x, y, w, h) = cv2.boundingRect(c)
122. cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
123. occupied = 1
124. text = "Occupied" if occupied == 1 else "UnOccupied"
125. cv2.putText(img, "Room status: {0}".format(
126. text), (15, 20+5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 0), 2)
127. cv2.putText(img, datetime.datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"),
128. (10, img.shape[0]-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.35, (0, 0, 255), 1)
129. return occupied, img
130. def get\_objectdetections(input\_img):
131. global object\_detector
132. '''
133. for eachObject in detections:
134. print(eachObject["name"] , " : ", eachObject["percentage\_probability"], " : ", eachObject["box\_points"] )
135. print("--------------------------------")
136. '''
137. # plt.imshow(returned\_image)
138. return object\_detector.detectObjectsFromImage(input\_type='array', input\_image=input\_img, output\_type='array', minimum\_percentage\_probability=85)
139. def give\_embedding(img\_):
140. #img\_ = cv2.resize(img\_, (256, 256), interpolation=cv2.INTER\_CUBIC)
141. # extract gray img
142. # gray\_img=np.eye(256)
143. # gray\_img=cv2.cvtColor(img.copy(),cv2.COLOR\_BGR2GRAY)
144. # Create a HOG face detector using the built-in dlib class
145. # print(image)
146. '''face=face\_detector(gray\_img,1)
147. if len(face)<1:
148. return []'''
149. # detected\_faces=face
150. # face=face[0]
151. # face\_img=cv2.rectangle(img,(detected\_faces[0].left(),detected\_faces[0].top()),(detected\_faces[0].right(),detected\_faces[0].bottom()),(0,255,0),1)
152. # face\_img=img[face.top():face.bottom(),face.left():face.right()]
153. #img=cv2.cvtColor(img\_,cv2.COLOR\_BGR2GRAY)
154. if img\_.ndim==3:
155. img=cv2.cvtColor(img\_,cv2.COLOR\_BGR2GRAY)
156. else:
157. img=img\_
158. img=cv2.resize(img,(256,256),interpolation=cv2.INTER\_CUBIC)
159. img=align\_face(img.astype(np.float32))
160. face\_embedding = np.array(face\_encodings(img[:,:,np.newaxis].repeat(3, 2),known\_face\_locations=[(0,img.shape[1],img.shape[0],0)])).ravel()
161. # Normalize???
162. '''from sklearn.preprocessing import StandardScaler
163. scaler = StandardScaler()
164. if len(face\_embedding) > 0:
165. face\_embedding = face\_embedding.reshape(128, 1)
166. face\_embedding = scaler.fit\_transform(face\_embedding)
167. print('Face Embedding', face\_embedding)
168. '''
169. return face\_embedding
170. def load\_face\_recog\_model\_DNN():
171. model = load\_model('model\_checkpoint\_face\_recog\_aligned.h5')
172. oneHot2Name = np.load('oneHot2Name.npy').item()
173. # Test Image
174. return model, oneHot2Name
175. model, oneHot2Name = load\_face\_recog\_model\_DNN()
176. def recognise\_img\_DNN(test\_img):
177. global oneHot2Name
178. global model
179. embd = give\_embedding(test\_img)
180. #print(embd)
181. if len(embd) < 1:
182. return 'unknown'
183. predicted = model.predict(np.array(embd).reshape(1, 128))
184. face = oneHot2Name[np.argmax(predicted[0])]
185. print("predicted: {0} ".format(face))
186. # cv2.imshow(oneHot2Name[predicted[0]],cv2.imread(test\_img))
187. # cv2.waitKey()
188. # cv2.destroyAllWindows()
189. return face
190. face\_match\_dir='D:/dataset/Image/Face Dataset custom'
191. def face\_match(test\_img):
192. found=0
193. test\_encoding=give\_embedding(test\_img)
194. for img\_ in listdir(face\_match\_dir):
195. img=load\_image\_file(face\_match\_dir+'/' + img\_)
196. current\_encoding=give\_embedding(img)
197. #print(current\_encoding.shape,test\_encoding.shape)
198. result=compare\_faces([test\_encoding], current\_encoding)
199. if(result[0])==True:
200. face=re.search('(.\*)\_.\*',img\_).group(1)
201. found=1
202. print('[FACE MATCH FOUND:]....',face)
203. break
204. if found==0:
205. face='unknown'
206. return face
207. '''
208. def load\_face\_recog\_model(train=False):
210. with open('SVMmodel.pickle', 'rb') as file:
211. model =loads(open('SVMmodel.pickle', "rb").read())
213. oneHot2Name=np.load('oneHot2Name.npy').item()
214. #Test Image
215. return model,oneHot2Name
216. model,oneHot2Name=load\_face\_recog\_model()
217. def recognise\_img\_SVM(test\_img):
218. global oneHot2Name
219. global model
220. embd=give\_embedding(test\_img)
221. if len(embd)<1:
222. return 'unknown'
223. predicted=model.predict(np.array(embd).reshape(1,128))
224. print("predicted: {0} ".format(oneHot2Name[predicted[0]]))
225. #cv2.imshow(oneHot2Name[predicted[0]],cv2.imread(test\_img))
226. #cv2.waitKey()
227. #cv2.destroyAllWindows()
228. return oneHot2Name[predicted[0]]
229. '''
230. '''
231. def extract\_save\_faces(img\_,trackers):
232. #global trackers
233. global saved\_faces
234. save\_path= path.join( getcwd(),"Extracted Faces")
235. print("save\_path: ",save\_path)
236. for fid in trackers.keys():
237. if fid not in saved\_faces:
238. tracked\_position=trackers[fid].get\_position()
239. t\_x = int(tracked\_position.left())
240. t\_y = int(tracked\_position.top())
241. t\_w = int(tracked\_position.width())
242. t\_h = int(tracked\_position.height())
243. print("Saving Face")
244. #print(img\_[t\_x:t\_x+t\_w][t\_y:t\_y+t\_h])
245. #print("Saving to: ", path.join(save\_path,'{0}.jpg'.format(len(saved\_faces)+1)))
246. cv2.imwrite( path.join(save\_path,'{0}.jpg'.format(len( listdir('Extracted Faces'))+1)),img\_[ t\_y:t\_y+t\_h,t\_x:t\_x+t\_w ])
247. saved\_faces.append(fid)'''
248. def tracker\_exist(x, y, w, h, trackers):
249. #global trackers
250. for fid in trackers.keys():
251. tracked\_position = trackers[fid][0].get\_position()
252. t\_x = int(tracked\_position.left())
253. t\_y = int(tracked\_position.top())
254. t\_w = int(tracked\_position.width())
255. t\_h = int(tracked\_position.height())
256. t\_center\_x = t\_x + 0.5\*t\_w
257. t\_center\_y = t\_y + 0.5\*t\_h
258. # check if the centerpoint of the face is within the
259. # rectangleof a tracker region. Also, the centerpoint
260. # of the tracker region must be within the region
261. # detected as a face. If both of these conditions hold
262. # we have a match
263. center\_x = x + 0.5\*w
264. center\_y = y + 0.5\*h
265. if (x <= t\_center\_x <= (x+w)) and (y <= t\_center\_y <= (y+h)) and (t\_x <= center\_x <= (t\_x+t\_w)) and (t\_y <= center\_y <= (t\_y+t\_h)):
266. return True
267. return False
268. def delete\_trackers(img, face\_count, trackers):
269. #global trackers
270. #global face\_count
271. fidsToDelete = []
272. for fid in trackers.keys():
273. track\_quality = trackers[fid][0].update(img)
274. if track\_quality < 9:
275. fidsToDelete.append(fid)
276. for fid in fidsToDelete:
277. print("Removing tracker " + str(fid) + " from list of trackers")
278. # trackers.pop(fid,None)
279. del trackers[fid]
280. # face\_count-=1 # as decrease the count and then there may be duplicate entries
281. # Don't need Haar detector, will detect using dlibs HOG
282. # facecascade=cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')
283. '''
284. eyecascade=cv2.CascadeClassifier('haarcascade\_eye.xml')
285. '''
286. #face\_detector = get\_frontal\_face\_detector()
287. '''
288. for video\_name in listdir('Videos'):
289. print("Video Name: ",video\_name)
290. vs=cv2.VideoCapture('Videos/'+video\_name)
291. '''
292. def show\_img(img):
293. #print('[INFO] Showing Image')
294. stop = 0
295. cv2.imshow('Security Feed', img)
296. # to make faster ENABLE to save output data to be used as training data
297. # extract\_save\_faces(orig\_img,trackers)
298. key = cv2.waitKey(1) & 0xFF
299. if key == ord('q'):
300. stop = 1
301. cv2.destroyAllWindows()
302. return stop
303. # load the tokenizer
304. tokenizer = load(open('tokenizer.pkl', 'rb'))
305. # pre-define the max sequence length (from training)
306. max\_length = 34
307. # load the model
308. caption\_model = load\_model('model\_18.h5')
309. # load and prepare the photograph
310. # extract features from each photo in the directory
311. def extract\_features(filename):
312. # load the model
313. model = inception\_resnet\_v2.InceptionResNetV2(
314. include\_top=True,
315. weights='imagenet',
316. input\_tensor=None,
317. input\_shape=None,
318. pooling=None,
319. classes=1000)
320. # re-structure the model
321. #model.layers.pop()
322. model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
323. # load the photo
324. #image = load\_img(filename, target\_size=(299, 299))
325. image = cv2.resize(filename,(299,299))
326. # convert the image pixels to a numpy array
327. image = img\_to\_array(image)
328. # reshape data for the model
329. image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
330. # prepare the image for the VGG model
331. image = inception\_resnet\_v2.preprocess\_input(image)
332. # get features
333. feature = model.predict(image, verbose=0)
334. return feature
335. # map an integer to a word
336. def word\_for\_id(integer, tokenizer):
337. for word, index in tokenizer.word\_index.items():
338. if index == integer:
339. return word
340. return None
341. # generate a description for an image
342. def generate\_desc(model, tokenizer, photo, max\_length):
343. # seed the generation process
344. in\_text = 'startseq'
345. # iterate over the whole length of the sequence
346. for i in range(max\_length):
347. # integer encode input sequence
348. sequence = tokenizer.texts\_to\_sequences([in\_text])[0]
349. # pad input
350. sequence = pad\_sequences([sequence], maxlen=max\_length)
351. # predict next word
352. yhat = caption\_model.predict([photo, sequence], verbose=0)
353. # convert probability to integer
354. yhat = np.argmax(yhat)
355. # map integer to word
356. word = word\_for\_id(yhat, tokenizer)
357. # stop if we cannot map the word
358. if word is None:
359. break
360. # append as input for generating the next word
361. in\_text += ' ' + word
362. # stop if we predict the end of the sequence
363. if word == 'endseq':
364. break
365. return in\_text
366. caption='Empty Caption'
367. caption\_threat\_check='No'
368. def get\_caption(img):
369. global caption
370. global caption\_threat\_check
371. feat = extract\_features(img)
372. # generate description
373. caption = generate\_desc(model, tokenizer,feat, max\_length)
374. caption\_threat\_check=caption\_classifier()
375. loaded\_model = load(open('threat\_model\_rfc', 'rb'))
376. vectorizer = load(open('vectorizer', 'rb'))
377. def caption\_classifier():
378. global caption
379. clean\_test\_reviews=[]
380. clean\_test\_reviews.append(" ".join(KaggleWord2VecUtility.review\_to\_wordlist(caption,True)))
381. test\_data\_features=vectorizer.transform(clean\_test\_reviews)
382. test\_data\_features=test\_data\_features.toarray()
383. threat=loaded\_model.predict(test\_data\_features)
384. return threat
385. from threading import Thread
386. stop = 0
387. counter=0
388. while not stop:
389. vs = cv2.VideoCapture(0)
390. vs.set(cv2.CAP\_PROP\_FPS, 12)
391. vs.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 500)
392. vs.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 500)
393. face\_count=0
394. # eye\_count=0
395. trackers = {}
396. \_, first\_frame = vs.read()
397. del \_
398. # print(first\_frame)
399. first\_frame = cv2.resize(first\_frame, (500, 500))
400. first\_frame = cv2.cvtColor(first\_frame, cv2.COLOR\_BGR2GRAY)
401. first\_frame = cv2.GaussianBlur(first\_frame, (21, 21), 0)
402. while True:
403. rc, img = vs.read()
404. '''for i in range(0,5):
405. x,y=vs.read()'''
406. #img = cv2.resize(img, (0,0), fx=0.5, fy=0.5)
407. # print(img)
408. motion, img = detect\_motion(first\_frame, img)
409. if(not motion):
410. stop = show\_img(img)
411. if(stop):
412. break
413. print('[MOTION NOT DETECTED].....................')
414. continue
415. print('[MOTION DETECTED].....................')
416. img, detections = get\_objectdetections(img)
417. print('[INFO] Got Objects Present in the Image')

420. if counter%40==10:
421. print('[Generating captions].....................')
422. get\_caption(img)
423. #t1 =Thread(target=get\_caption, args=(img,))
424. #t1.start()
425. print('[CAPTION] GENERATED...{0}'.format(caption))
426. #threat\_caption=caption\_classifier(caption)
427. print('[THREAT CLASSIFIER] ...{0}'.format(caption\_threat\_check))
428. # orig\_img=img.copy()q
429. counter+=1
431. cv2.putText(img,"Caption: {0}".format(caption), (15, 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)
432. cv2.putText(img,"Threat: {0}".format(str(caption\_threat\_check)), (15, 40), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)
433. '''for i in range(0,3):
434. rc,img=vs.read()
435. continue'''
436. #img = cv2.rotateImage(img, 90)
437. # print(img)
438. # cv2.imshow('Detector',img)
439. print('[INFO] Preprocessed Image From VideoStream')
440. delete\_trackers(img, face\_count, trackers)
441. print('[INFO] Deleted Trackers')
443. # faces=facecascade.detectMultiScale(gray\_img,1.3,5)
444. '''
445. eyes=eyecascade.detectMultiScale(gray\_img,1.3,5)'''
446. l=[d['name'] for d in detections]
447. if 'person' in l:
448. gray\_img = cv2.cvtColor(img.copy(), cv2.COLOR\_BGR2GRAY)
449. faces\_rect = face\_detector(gray\_img, 1)
450. print('[INFO] Detected Faces from the image')
451. # detected\_faces=face
452. #CREATE and ADD trackers for new faces
453. x = 0
454. y = 0
455. w = 0
456. h = 0
457. max\_area = 0
458. faces = []
459. for i in faces\_rect:
460. faces.append((i.left(), i.top(), i.right() -i.left(), i.bottom()-i.top()))
461. for (\_x, \_y, \_w, \_h) in faces:
462. if \_w\*\_h > max\_area:
463. x = \_x
464. y = \_y
465. h = \_h
466. w = \_w
467. max\_area = \_w\*\_h
468. if not tracker\_exist(x, y, h, w, trackers):
469. t1 = correlation\_tracker()
470. t1.start\_track(img, rectangle(int(x), int(y), int(x+w), int(y+h)))
471. cv2.putText(img, "New Face detected", (x+5, y+5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)
472. cv2.rectangle(img, (x-10, y-20),(x+w+10, y+h+20), (0, 0, 255), 1)
473. #face\_img=img[y-5:y+h+5, x-5:x+w+5]
474. face\_img = gray\_img[y-5:y+h+5, x-5:x+w+5]
475. '''personface1 = recognise\_img\_DNN(face\_img)
476. personface2=face\_match(face\_img)
477. if personface1==personface2:
478. personface=personface1
479. else:
480. personface='unknown'''
481. personface = recognise\_img\_DNN(face\_img)
482. #emotion\_img = cv2.cvtColor(face\_img.copy(), cv2.COLOR\_BGR2GRAY)
483. emotion\_img = cv2.resize(face\_img, (256, 256), interpolation=cv2.INTER\_CUBIC)
484. emotion = detect\_emotion(emotion\_img)
485. print('[ADDING NEW FACE] '+str(personface))
486. trackers[face\_count] = [t1, personface, emotion]
487. face\_count += 1
488. print('[INFO] Added New\_faces to tracker list')
489. #ADD rectangles and text to existing trackers
490. for fid in trackers.keys():
491. tracked\_position = trackers[fid][0].get\_position()
492. t\_x = int(tracked\_position.left())
493. t\_y = int(tracked\_position.top())
494. t\_w = int(tracked\_position.width())
495. t\_h = int(tracked\_position.height())
496. # face\_img=cv2.rectangle(img,(detected\_faces[0].left(),detected\_faces[0].top()),(detected\_faces[0].right(),detected\_faces[0].bottom()),(0,255,0),1)
497. #face\_img = img[t\_y-5:t\_y+t\_h+5, t\_x-5:t\_x+t\_w+5]
498. # face\_embedding=np.array(face\_recognition.face\_encodings(face\_img)).ravel()
499. recog\_face = trackers[fid][1]
500. emotion = trackers[fid][2]
501. cv2.rectangle(img, (t\_x, t\_y),
502. (t\_x + t\_w, t\_y + t\_h),
503. (255, 0, 0), 1)
504. cv2.putText(img, "{0}".format(recog\_face), (t\_x+5, t\_y-15),
505. cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 1)
506. cv2.putText(img, "{0}".format(emotion), (t\_x+5, t\_y+t\_h),
507. cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 1)
508. print('[INFO] Added Rectangles and Text for Existing Trackers')
509. '''
510. x1=0
511. y1=0
512. w1=0
513. h1=0
514. max\_area1=0
515. for (\_x,\_y,\_w,\_h) in eyes:
516. if \_w\*\_h>max\_area1:
517. x1=\_x
518. y1=\_y
519. h1=\_h
520. w1=\_w
521. max\_area1=\_w\*\_h
523. cv2.rectangle(img, (x1-5,y1-10), (x1+w1+5, y1+h1+10), (0,255,0),1)
524. '''
526. stop = show\_img(img)
527. '''if counter%15==0:
528. t1.join()'''
529. if(stop):
530. break
531. vs.release()
532. cv2.destroyAllWindows()

2) VIDEO MODULE