**Module 10**

Use Yolov4 to detect people in a video,use the yolov4 pretrained weights.

import numpy as np

import cv2

#get the saved video file as stream

file\_video\_stream = cv2.VideoCapture('E:/ARTIFICIAL INTELIGENCE/Class/Object Detector,Image Segmentation,Model optimisation and Inference/video (1).mp4')

#create a while loop

while True:

#get the current frame from video stream

ret,current\_frame = file\_video\_stream.read()

#use the video current frame instead of image

img\_to\_detect = current\_frame

img\_height = img\_to\_detect.shape[0]

img\_width = img\_to\_detect.shape[1]

# convert to blob to pass into model

img\_blob = cv2.dnn.blobFromImage(img\_to\_detect, 0.003922, (416, 416), swapRB=True, crop=False)

#recommended by yolo authors, scale factor is 0.003922=1/255, width,height of blob is 320,320

#accepted sizes are 320×320,416×416,608×608. More size means more accuracy but less speed

# set of 80 class labels

class\_labels = ["person","bicycle","car","motorcycle","airplane","bus","train","truck","boat",

"trafficlight","firehydrant","stopsign","parkingmeter","bench","bird","cat",

"dog","horse","sheep","cow","elephant","bear","zebra","giraffe","backpack",

"umbrella","handbag","tie","suitcase","frisbee","skis","snowboard","sportsball",

"kite","baseballbat","baseballglove","skateboard","surfboard","tennisracket",

"bottle","wineglass","cup","fork","knife","spoon","bowl","banana","apple",

"sandwich","orange","broccoli","carrot","hotdog","pizza","donut","cake","chair",

"sofa","pottedplant","bed","diningtable","toilet","tvmonitor","laptop","mouse",

"remote","keyboard","cellphone","microwave","oven","toaster","sink","refrigerator",

"book","clock","vase","scissors","teddybear","hairdrier","toothbrush"]

#Declare List of colors as an array

#Green, Blue, Red, cyan, yellow, purple

#Split based on ',' and for every split, change type to int

#convert that to a numpy array to apply color mask to the image numpy array

class\_colors = ["0,255,0","0,0,255","255,0,0","255,255,0","0,255,255"]

class\_colors = [np.array(every\_color.split(",")).astype("int") for every\_color in class\_colors]

class\_colors = np.array(class\_colors)

class\_colors = np.tile(class\_colors,(16,1))

# Loading pretrained model

# input preprocessed blob into model and pass through the model

# obtain the detection predictions by the model using forward() method

yolo\_model = cv2.dnn.readNetFromDarknet('E:/ARTIFICIAL INTELIGENCE/Assignment/Object Detector,Image Segmentation,Model optimisation and Inference/yolov4.cfg','E:/ARTIFICIAL INTELIGENCE/Assignment/Object Detector,Image Segmentation,Model optimisation and Inference/yolov4.weights')

# Get all layers from the yolo network

# Loop and find the last layer (output layer) of the yolo network

yolo\_layers = yolo\_model.getLayerNames()

yolo\_output\_layer = [yolo\_layers[yolo\_layer[0] - 1] for yolo\_layer in yolo\_model.getUnconnectedOutLayers()]

# input preprocessed blob into model and pass through the model

yolo\_model.setInput(img\_blob)

# obtain the detection layers by forwarding through till the output layer

obj\_detection\_layers = yolo\_model.forward(yolo\_output\_layer)

############## NMS Change 1 ###############

# initialization for non-max suppression (NMS)

# declare list for [class id], [box center, width & height[], [confidences]

class\_ids\_list = []

boxes\_list = []

confidences\_list = []

############## NMS Change 1 END ###########

# loop over each of the layer outputs

for object\_detection\_layer in obj\_detection\_layers:

# loop over the detections

for object\_detection in object\_detection\_layer:

# obj\_detections[1 to 4] => will have the two center points, box width and box height

# obj\_detections[5] => will have scores for all objects within bounding box

all\_scores = object\_detection[5:]

predicted\_class\_id = np.argmax(all\_scores)

prediction\_confidence = all\_scores[predicted\_class\_id]

# take only predictions with confidence more than 50%

if prediction\_confidence > 0.50:

#obtain the bounding box co-oridnates for actual image from resized image size

bounding\_box = object\_detection[0:4] \* np.array([img\_width, img\_height, img\_width, img\_height])

(box\_center\_x\_pt, box\_center\_y\_pt, box\_width, box\_height) = bounding\_box.astype("int")

start\_x\_pt = int(box\_center\_x\_pt - (box\_width / 2))

start\_y\_pt = int(box\_center\_y\_pt - (box\_height / 2))

############## NMS Change 2 ###############

#save class id, start x, y, width & height, confidences in a list for nms processing

#make sure to pass confidence as float and width and height as integers

class\_ids\_list.append(predicted\_class\_id)

confidences\_list.append(float(prediction\_confidence))

boxes\_list.append([start\_x\_pt, start\_y\_pt, int(box\_width), int(box\_height)])

############## NMS Change 2 END ###########

############## NMS Change 3 ###############

# Applying the NMS will return only the selected max value ids while suppressing the non maximum (weak) overlapping bounding boxes

# Non-Maxima Suppression confidence set as 0.5 & max\_suppression threhold for NMS as 0.4 (adjust and try for better perfomance)

max\_value\_ids = cv2.dnn.NMSBoxes(boxes\_list, confidences\_list, 0.5, 0.4)

# loop through the final set of detections remaining after NMS and draw bounding box and write text

for max\_valueid in max\_value\_ids:

max\_class\_id = max\_valueid[0]

box = boxes\_list[max\_class\_id]

start\_x\_pt = box[0]

start\_y\_pt = box[1]

box\_width = box[2]

box\_height = box[3]

#get the predicted class id and label

predicted\_class\_id = class\_ids\_list[max\_class\_id]

predicted\_class\_label = class\_labels[predicted\_class\_id]

prediction\_confidence = confidences\_list[max\_class\_id]

############## NMS Change 3 END ###########

#obtain the bounding box end co-oridnates

end\_x\_pt = start\_x\_pt + box\_width

end\_y\_pt = start\_y\_pt + box\_height

#get a random mask color from the numpy array of colors

box\_color = class\_colors[predicted\_class\_id]

#convert the color numpy array as a list and apply to text and box

box\_color = [int(c) for c in box\_color]

# print the prediction in console

predicted\_class\_label = "{}: {:.2f}%".format(predicted\_class\_label, prediction\_confidence \* 100)

print("predicted object {}".format(predicted\_class\_label))

# draw rectangle and text in the image

cv2.rectangle(img\_to\_detect, (start\_x\_pt, start\_y\_pt), (end\_x\_pt, end\_y\_pt), box\_color, 1)

cv2.putText(img\_to\_detect, predicted\_class\_label, (start\_x\_pt, start\_y\_pt-5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, box\_color, 1)

cv2.imshow("Detection Output", img\_to\_detect)

#terminate while loop if 'q' key is pressed

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

break

#releasing the stream

#close all opencv windows

file\_video\_stream.release()

cv2.destroyAllWindows()

Note : Download yolov4 weights .