Network and function

[1]:

**1 Detecting social Distance**

Network

Importing the libraries

**import backbone**

**import tensorflow as tf import cv2**

**import numpy as np import cv2**

**import numpy as np**

**from scipy.spatial.distance import** pdist, squareform

**import cv2 import os import argparse**

**from network\_model import** model

**from aux\_functions import** \*

D:\anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint8 = np.dtype([("qint8", np.int8, 1)])

D:\anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_quint8 = np.dtype([("quint8", np.uint8, 1)])

D:\anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:528: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint16 = np.dtype([("qint16", np.int16, 1)])

D:\anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:529: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_quint16 = np.dtype([("quint16", np.uint16, 1)])

D:\anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:530: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint32 = np.dtype([("qint32", np.int32, 1)])

D:\anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

np\_resource = np.dtype([("resource", np.ubyte, 1)])

Make Network

1. : **class model**:

**def** init (self):

detection\_graph, self.category\_index = backbone.set\_model( "ssd\_mobilenet\_v1\_coco\_2018\_01\_28", "mscoco\_label\_map.pbtxt"

)

self.sess = tf.InteractiveSession(graph=detection\_graph) self.image\_tensor = detection\_graph.get\_tensor\_by\_name("image\_tensor:0") self.detection\_boxes = detection\_graph.

*‹→*get\_tensor\_by\_name("detection\_boxes:0")

self.detection\_scores = detection\_graph.

*‹→*get\_tensor\_by\_name("detection\_scores:0")

self.detection\_classes = detection\_graph.get\_tensor\_by\_name( "detection\_classes:0"

)

self.num\_detections = detection\_graph.

*‹→*get\_tensor\_by\_name("num\_detections:0")

**def** get\_category\_index(self):

**return** self.category\_index

**def** detect\_pedestrians(self, frame): input\_frame = frame

image\_np\_expanded = np.expand\_dims(input\_frame, axis=0) (boxes, scores, classes, num) = self.sess.run(

[

self.detection\_boxes, self.detection\_scores, self.detection\_classes, self.num\_detections,

],

feed\_dict={self.image\_tensor: image\_np\_expanded},

)

classes = np.squeeze(classes).astype(np.int32) boxes = np.squeeze(boxes)

scores = np.squeeze(scores)

pedestrian\_score\_threshold = 0.35 pedestrian\_boxes = [] total\_pedestrians = 0

**for** i **in** range(int(num[0])):

**if** classes[i] **in** self.category\_index.keys():

class\_name = self.category\_index[classes[i]]["name"]

*# print(class\_name)*

**if** class\_name == "person" **and** scores[i] >␣

*‹→*pedestrian\_score\_threshold:

total\_pedestrians += 1 score\_pedestrian = scores[i] pedestrian\_boxes.append(boxes[i])

**return** pedestrian\_boxes, total\_pedestrians

Make the function

1. : **def** plot\_lines\_between\_nodes(warped\_points, bird\_image, d\_thresh): p = np.array(warped\_points)

dist\_condensed = pdist(p)

dist = squareform(dist\_condensed)

dd = np.where(dist < d\_thresh \* 6 / 10) close\_p = []

color\_10 = (96,160,48)

lineThickness = 4

ten\_feet\_violations = len(np.where(dist\_condensed < 10 / 6 \* d\_thresh)[0])

**for** i **in** range(int(np.ceil(len(dd[0]) / 2))):

**if** dd[0][i] != dd[1][i]: point1 = dd[0][i] point2 = dd[1][i]

close\_p.append([point1, point2]) cv2.line(

bird\_image,

(p[point1][0], p[point1][1]),

(p[point2][0], p[point2][1]), color\_10,

lineThickness,

)

dd = np.where(dist < d\_thresh)

six\_feet\_violations = len(np.where(dist\_condensed < d\_thresh)[0]) total\_pairs = len(dist\_condensed)

danger\_p = []

color\_6 = (96,160,48)

**for** i **in** range(int(np.ceil(len(dd[0]) / 2))):

**if** dd[0][i] != dd[1][i]: point1 = dd[0][i] point2 = dd[1][i]

danger\_p.append([point1, point2]) cv2.line(

bird\_image,

(p[point1][0], p[point1][1]),

(p[point2][0], p[point2][1]), color\_6,

lineThickness,

)

*# Display Birdeye view*

cv2.imshow("Bird Eye View", bird\_image) cv2.waitKey(1)

**return** six\_feet\_violations, ten\_feet\_violations, total\_pairs

**def** plot\_points\_on\_bird\_eye\_view(frame, pedestrian\_boxes, M, scale\_w, scale\_h): frame\_h = frame.shape[0]

frame\_w = frame.shape[1] node\_radius = 10

color\_node = (96,160,48) *#96,160,48*

thickness\_node = 20

solid\_back\_color = (96,160,48) *#41, 41, 41*

blank\_image = np.zeros(

(int(frame\_h \* scale\_h), int(frame\_w \* scale\_w), 3), np.uint8

)

blank\_image[:] = solid\_back\_color warped\_pts = []

**for** i **in** range(len(pedestrian\_boxes)):

mid\_point\_x = int(

(pedestrian\_boxes[i][1] \* frame\_w + pedestrian\_boxes[i][3] \*␣

*‹→*frame\_w) / 2

)

mid\_point\_y = int(

(pedestrian\_boxes[i][0] \* frame\_h + pedestrian\_boxes[i][2] \*␣

*‹→*frame\_h) / 2

)

pts = np.array([[[mid\_point\_x, mid\_point\_y]]], dtype="float32") warped\_pt = cv2.perspectiveTransform(pts, M)[0][0]

warped\_pt\_scaled = [int(warped\_pt[0] \* scale\_w), int(warped\_pt[1] \*␣

*‹→*scale\_h)]

warped\_pts.append(warped\_pt\_scaled) bird\_image = cv2.circle(

blank\_image,

(warped\_pt\_scaled[0], warped\_pt\_scaled[1]), node\_radius,

color\_node, thickness\_node,

)

**return** warped\_pts, bird\_image

[4]:

**def** get\_camera\_perspective(img, src\_points): IMAGE\_H = img.shape[0]

IMAGE\_W = img.shape[1]

src = np.float32(np.array(src\_points))

dst = np.float32([[0, IMAGE\_H], [IMAGE\_W, IMAGE\_H], [0, 0], [IMAGE\_W, 0]])

M = cv2.getPerspectiveTransform(src, dst) M\_inv = cv2.getPerspectiveTransform(dst, src)

**return** M, M\_inv

[5]:

**def** put\_text(frame, text, text\_offset\_y=25): font\_scale = 0.8

font = cv2.FONT\_HERSHEY\_SIMPLEX

rectangle\_bgr = (35, 35, 35)

(text\_width, text\_height) = cv2.getTextSize( text, font, fontScale=font\_scale, thickness=1

)[0]

*# set the text start position*

text\_offset\_x = frame.shape[1] - 400

*# make the coords of the box with a small padding of two pixels*

box\_coords = (

(text\_offset\_x, text\_offset\_y + 5),

(text\_offset\_x + text\_width + 2, text\_offset\_y - text\_height - 2),

)

frame = cv2.rectangle(

frame, box\_coords[0], box\_coords[1], rectangle\_bgr, cv2.FILLED

)

frame = cv2.putText( frame,

text,

(text\_offset\_x, text\_offset\_y), font,

fontScale=font\_scale, color=(96,160,48), *#255, 255, 255*

thickness=1,

)

**return** frame, 2 \* text\_height + text\_offset\_y

[6]:

**def** calculate\_stay\_at\_home\_index(total\_pedestrians\_detected, frame\_num, fps): normally\_people = 10

pedestrian\_per\_sec = np.round(total\_pedestrians\_detected / frame\_num, 1) sh\_index = 1 - pedestrian\_per\_sec / normally\_people

**return** pedestrian\_per\_sec, sh\_index

[2]:

**def** plot\_pedestrian\_boxes\_on\_image(frame, pedestrian\_boxes): frame\_h = frame.shape[0]

frame\_w = frame.shape[1] thickness = 2

*# color\_node = (80, 172, 110)*

color\_node = (96,160,48)

*# color\_10 = (160, 48, 112)*

**for** i **in** range(len(pedestrian\_boxes)): pt1 = (

int(pedestrian\_boxes[i][1] \* frame\_w), int(pedestrian\_boxes[i][0] \* frame\_h),

)

pt2 = (

int(pedestrian\_boxes[i][3] \* frame\_w), int(pedestrian\_boxes[i][2] \* frame\_h),

)

frame\_with\_boxes = cv2.rectangle(frame, pt1, pt2, color\_node, thickness)

**return** frame\_with\_boxes

[ ]:

[ ]: