Network and function

May 8, 2020

1 Detecting social Distance

Network

Importing the libraries

[1]:
```
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

[2]:
```
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

```
[3]: 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
```

[ ]:

[ ]: