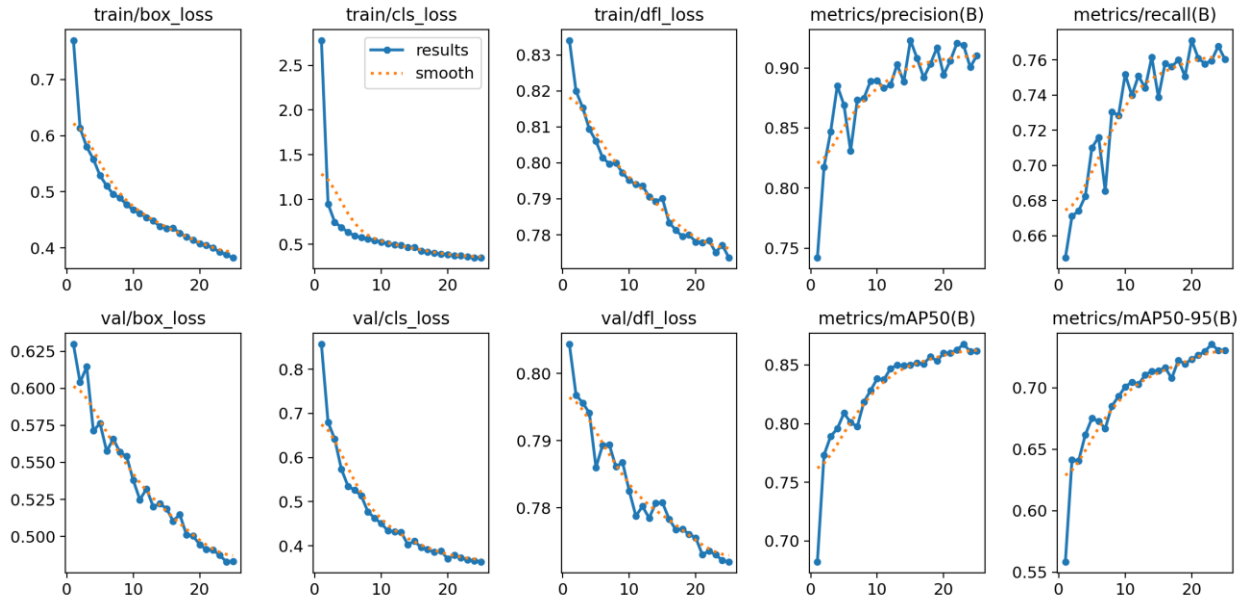
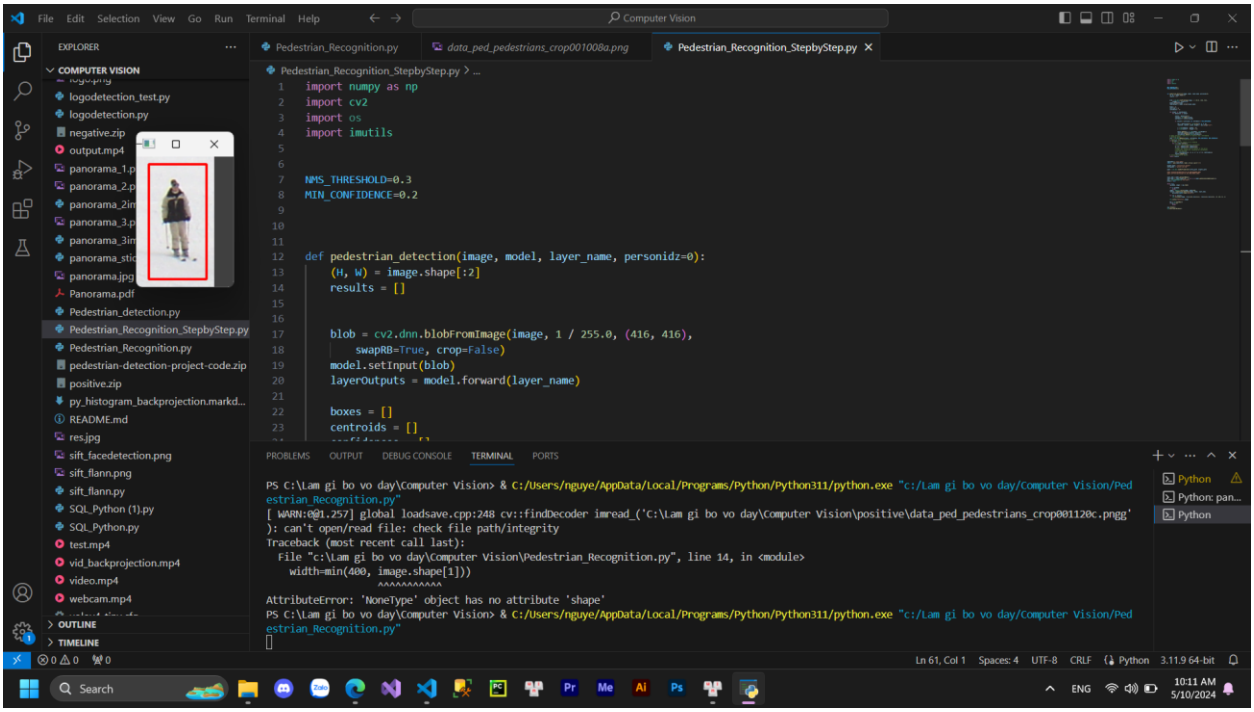


Assign 1

```
import numpy as np
import cv2
import os
import imutils


NMS_THRESHOLD=0.3
MIN_CONFIDENCE=0.2



def pedestrian_detection(image, model, layer_name, personidz=0):
    (H, W) = image.shape[:2]
    results = []
```

```
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
    model.setInput(blob)
    layerOutputs = model.forward(layer_name)

    boxes = []
    centroids = []
    confidences = []

    for output in layerOutputs:
        for detection in output:

            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            if classID == personidz and confidence > MIN_CONFIDENCE:

                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                boxes.append([x, y, int(width), int(height)])
                centroids.append((centerX, centerY))
                confidences.append(float(confidence))
    # apply non-maxima suppression to suppress weak, overlapping
    # bounding boxes
    idzs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONFIDENCE, NMS_THRESHOLD)
    # ensure at least one detection exists
    if len(idzs) > 0:
        # loop over the indexes we are keeping
        for i in idzs.flatten():
            # extract the bounding box coordinates
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])
            # update our results list to consist of the person
            # prediction probability, bounding box coordinates,
            # and the centroid
            res = (confidences[i], (x, y, x + w, y + h), centroids[i])
            results.append(res)
    # return the list of results
    return results
```

```
labelsPath = "coco.names"
LABELS = open(labelsPath).read().strip().split("\n")

weights_path = "yolov4-tiny.weights"
config_path = "yolov4-tiny.cfg"

model = cv2.dnn.readNetFromDarknet(config_path, weights_path)
'''
model.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
model.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
'''

layer_name = model.getLayerNames()
layer_name = [layer_name[i[0] - 1] for i in model.getUnconnectedOutLayers()]
cap = cv2.VideoCapture("test.mp4")
writer = None

while True:
    (grabbed, image) = cap.read()

    if not grabbed:
        break
    image = imutils.resize(image, width=700)
    results = pedestrian_detection(image, model, layer_name,
        personidz=LABELS.index("person"))

    for res in results:
        cv2.rectangle(image, (res[1][0],res[1][1]), (res[1][2],res[1][3]), (0,
255, 0), 2)

    cv2.imshow("Detection",image)

    key = cv2.waitKey(1)
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()
```



Assign 2

```
import cv2
import imutils

# Initializing the HOG person
# detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

# Reading the Image
image = cv2.imread('C:\Lam gi bo vo day\Computer
Vision\positive\data_ped_pedestrians_crop001008a.png')

# Resizing the Image
image = imutils.resize(image,
                       width=min(400, image.shape[1]))

# Detecting all the regions in the
# Image that has a pedestrians inside it
(regions, _) = hog.detectMultiScale(image,
                                    winStride=(4, 4),
                                    padding=(4, 4),
```

```
                                    scale=1.05)

# Drawing the regions in the Image
for (x, y, w, h) in regions:
    cv2.rectangle(image, (x, y),
                  (x + w, y + h),
                  (0, 0, 255), 2)

# Showing the output Image
cv2.imshow("Image", image)
cv2.waitKey(0)

cv2.destroyAllWindows()
```