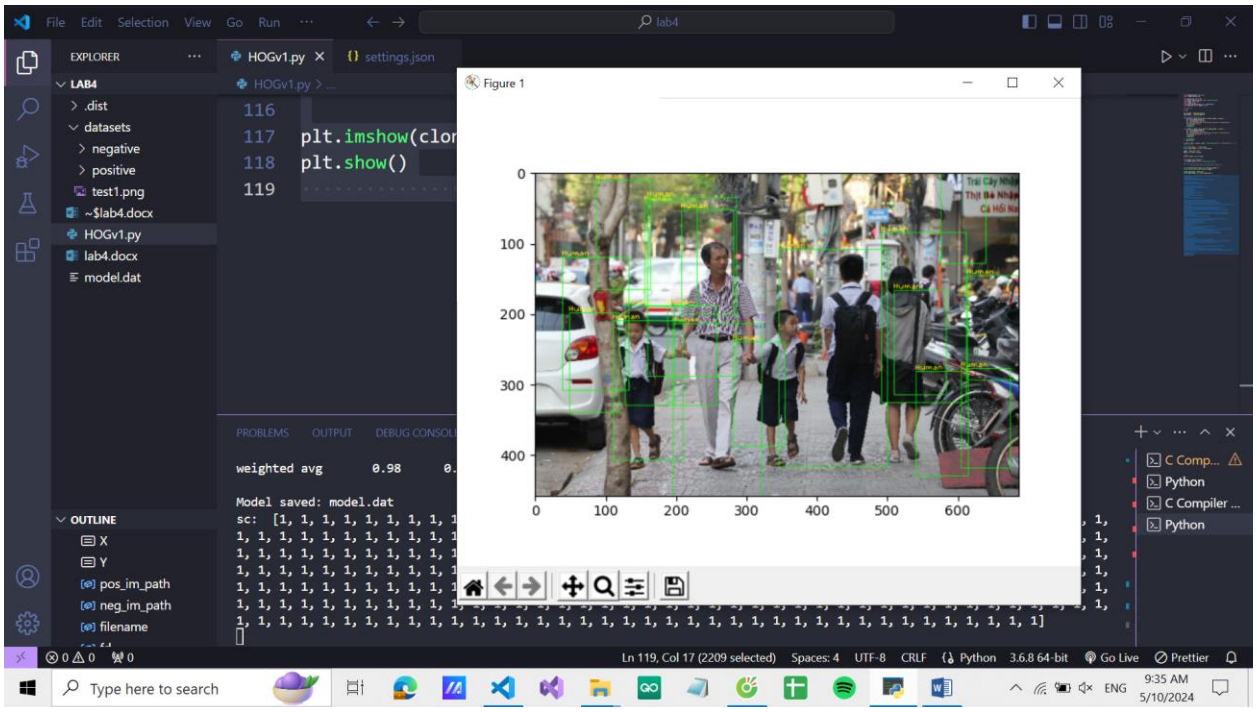
Nguyễn Lê Minh Khuê

2151013042

**Assignment 1:** pedestrian recognition by manually (step by step) (1 point)

**Result:**



```
plt.imshow(clon
plt.show()
```

```
weighted avg       0.98       0.

Model saved: model.dat
sc:  [1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

**Code:**

```python
from skimage.feature import hog
import joblib,glob,os,cv2

from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import svm
import numpy as np
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot as plt
```

```
#TRAINING SVM
X = []
Y = []

pos_im_path = 'datasets/positive'
neg_im_path = 'datasets/negative'

#Load the positive features
for filename in glob.glob(os.path.join(pos_im_path,
"*.png")):
    fd = cv2.imread(filename,0)
    fd = cv2.resize(fd, (64,128))
    fd = hog(fd,orientations=9, pixels_per_cell=(8,8),
visualize=False,cells_per_block=(2,2))
    X.append(fd)
    Y.append(1)

for filename in glob.glob(os.path.join(neg_im_path,
"*.png")):
    fd = cv2.imread(filename,0)
    fd = cv2.resize(fd, (64,128))
    fd = hog(fd,orientations=9, pixels_per_cell=(8,8),
visualize=False,cells_per_block=(2,2)) #9 huong, 8x8
    X.append(fd)
    Y.append(0)

X = np.float32(X)
Y = np.array(Y)

x_train, x_test, y_train, y_test = train_test_split(X,
Y, test_size=0.2) #lay 20 test, 80 train
```

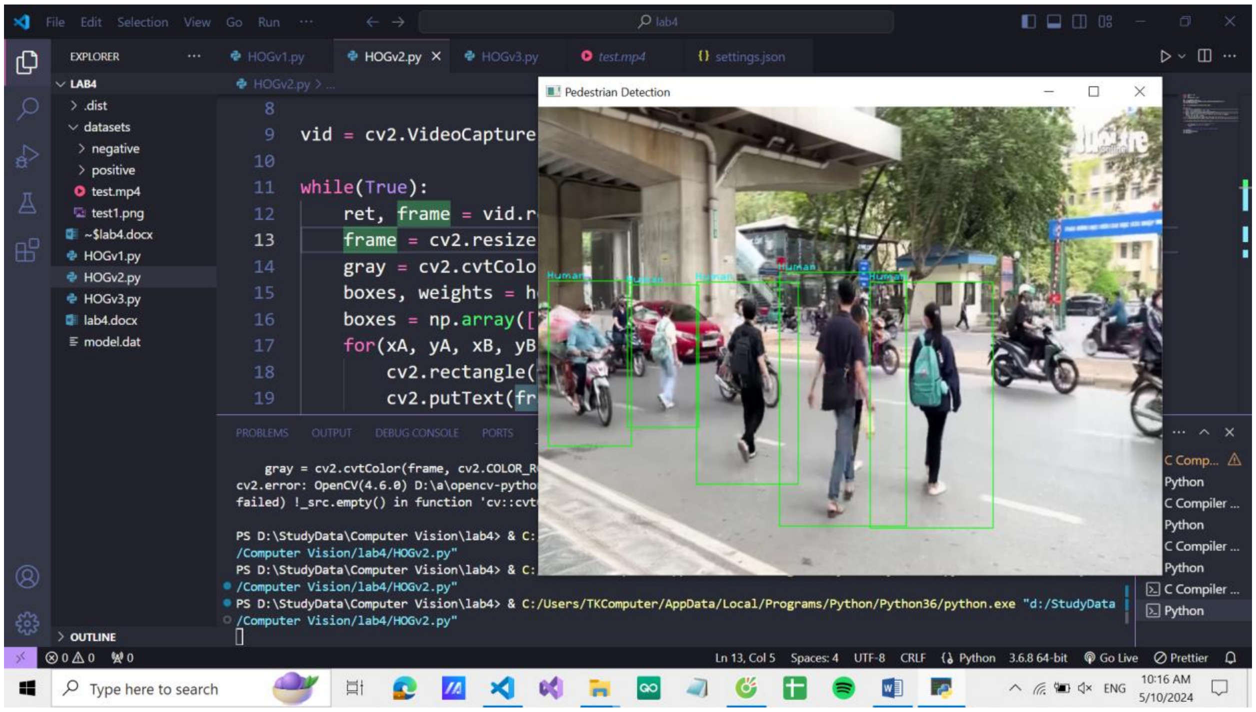
```python
print('Train data: ', len(x_train))
print('Train Labels: (1, 0)', len(y_train))

model = SVC(kernel='rbf')
model.fit(x_train, y_train)

#predict
y_pred = model.predict(x_test)

#confusion matric and accurancy
from sklearn import metrics
from sklearn.metrics import classification_report

#print(f"Classification report for classifier (model:
\n)"
#      f"(metrics.classification_report(y_test, y_pred):
\n")

print("Classification report for classifier (model):")
print(classification_report(y_test, y_pred))

joblib.dump(model, 'model.dat')
print('Model saved: {}'.format('model.dat'))

from imutils.object_detection import non_max_suppression
import imutils
from skimage import color
from skimage.transform import pyramid_gaussian

modelFile = 'model.dat'
inputFile = 'datasets/test1.png'
```

```python
image = cv2.imread(inputFile)
image = cv2.resize(image, (687, 458))
size = (64, 128)
step_size = (9, 9)
downscale = 1.05
#List to store the detection
detection = []
#The current scale off the image
scale = 0
model = joblib.load(modelFile)

def sliding_window(image, window_size, step_size):
    for y in range(0, image.shape[0], step_size[1]):
        for x in range(0, image.shape[1], step_size[0]):
            yield(x, y, image[y: y + window_size[1], x:
x + window_size[0]])

for im_scaled in pyramid_gaussian(image,
downscale=downscale):
    #The lisst contains detection at the current scale
    if im_scaled.shape[0] < size[1] or
im_scaled.shape[1] < size[0]:
        break
    for (x, y, window) in sliding_window(im_scaled,
size, step_size):
        if window.shape[0] != size[1] or window.shape[1]
!= size[0]:
            continue
        window = color.rgb2gray(window)
        fd = hog (window, orientations=9,
pixels_per_cell=(8,8), visualize=False,
cells_per_block=(2,2))
        fd = fd.reshape(1, -1)
        pred = model.predict(fd)
        if pred == 1:

            if model.decision_function(fd) > 0.5:
                detection.append(
                    (int(x * (downscale ** scale)),
int(y * (downscale ** scale)), model.predict(fd),
                    int(size[0] * (downscale ** scale)),
                    int(size[1] * (downscale **
scale))))

    scale += 1

clone = image.copy()
clone = cv2.cvtColor(clone, cv2.COLOR_BGR2RGB)
rects = np.array([[x, y, x + w, y +h] for [x, y, _, w,
h] in detection])
sc = [score[0] for (x, y, score, w, h) in detection]
print ("sc: ", sc)
sc = np.array(sc)
pick = non_max_suppression(rects, probs=sc,
overlapThresh=0.45)
for (x1, y1, x2, y2) in pick:
    cv2.rectangle(clone, (x1, y1), (x2, y2), (0, 255,
0))
    cv2.putText(clone, 'Human', (x1 - 2, y1 - 2), 1,
0.75, (255, 255, 0), 1)
```

```
plt.imshow(clone)
plt.show()
```

**Assignment 2:** detect pedestrians in image/video (0.5)

**Result:**



```
vid = cv2.VideoCapture

while(True):
    ret, frame = vid.r
    frame = cv2.resize
    gray = cv2.cvtColo
    boxes, weights = h
    boxes = np.array([
    for(xA, yA, xB, yB
        cv2.rectangle(
        cv2.putText(fr
```

```
    gray = cv2.cvtColor(frame, cv2.COLOR_R
cv2.error: OpenCV(4.6.0) D:\a\opencv-pytho
failed) !_src.empty() in function 'cv::cvt

PS D:\StudyData\Computer Vision\lab4> & C:
/Computer Vision/lab4/HOGv2.py"
PS D:\StudyData\Computer Vision\lab4> & C:
/Computer Vision/lab4/HOGv2.py"
PS D:\StudyData\Computer Vision\lab4> & C:/Users/TKComputer/AppData/Local/Programs/Python/Python36/python.exe "d:/StudyData
/Computer Vision/lab4/HOGv2.py"
```

**Code:**

```
import numpy as np
import cv2
import matplotlib as plt

hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDet
ector())
cv2.startWindowThread()

vid = cv2.VideoCapture('datasets/test.mp4')
```

```python
while(True):
    ret, frame = vid.read()
    frame = cv2.resize(frame, (640, 480))
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    boxes, weights = hog.detectMultiScale(frame,
winStride = (8, 8))
    boxes = np.array([[x, y, x + w, y + h] for (x, y, w,
h) in boxes])
    for(xA, yA, xB, yB) in boxes:
        cv2.rectangle(frame, (xA, yA), (xB, yB),
(0,  255, 0))
        cv2.putText(frame, 'Human', (xA - 2, yA - 2), 1,
0.75, (255, 255, 0), 1)

    cv2.imshow("Pedestrian Detection", frame)

        # Check for the 'q' key to quit the loop
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

    # Release the capture and close all OpenCV windows
vid.release()
cv2.destroyAllWindows()
cv2.waitKey(1)
```