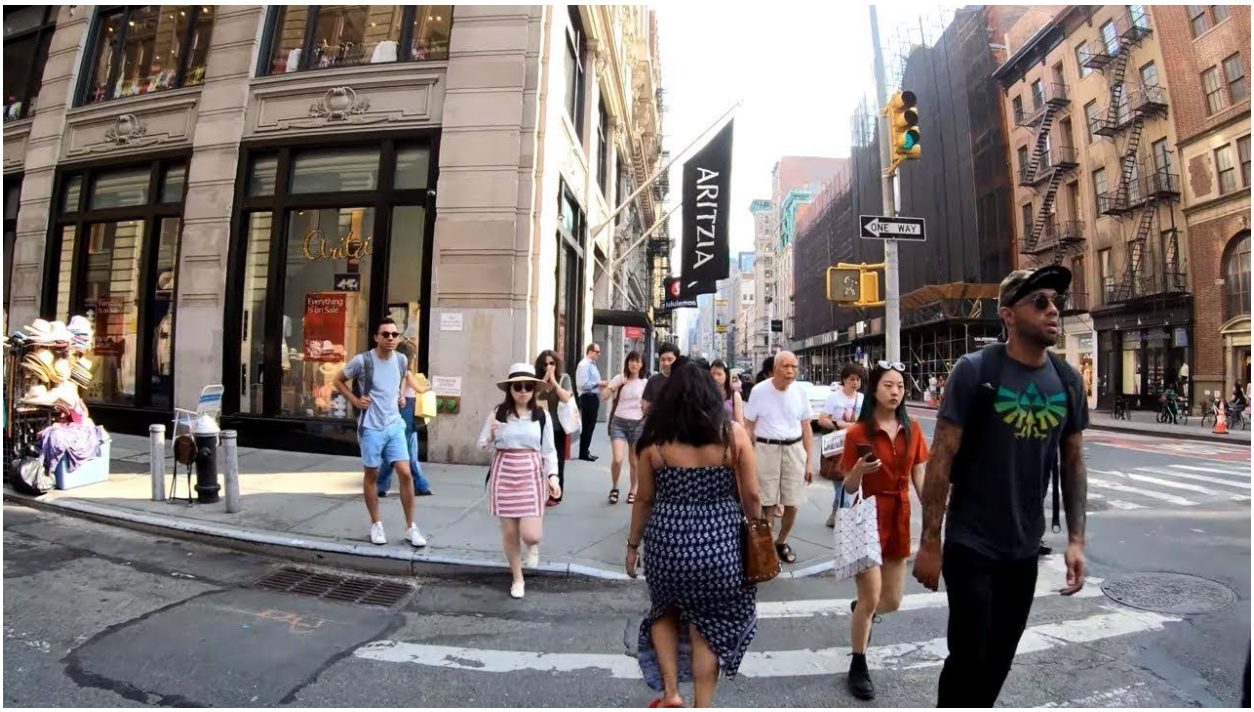
Tên : Nguyễn Sỹ Hà MSSV : 2151013018

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

Image input:



Output :

```
TERMINAL

Classification report for classifier LinearSVC():
              precision    recall  f1-score   support

           0       0.83      0.93      0.88        27
           1       0.93      0.85      0.89        33

    accuracy                           0.88        60
   macro avg       0.88      0.89      0.88        60
weighted avg       0.89      0.88      0.88        60


Model saved: models.dat
```

Src : Assignment 1

```python
from skimage.feature import hog
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot as plt

import joblib,glob,os,cv2
import numpy as np
```

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

pos_im_path = r'/Users/syha/Downloads/positive'
neg_im_path = r'/Users/syha/Downloads/negative'

#Load the pos 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)

#Load the neg features
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))
    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)
print('Train Data:',len(X_train))
print('Train Labels (1,0)', len(Y_train))

model = LinearSVC()
model.fit(X_train,Y_train)

#predict
Y_pred = model.predict(X_test)

#confusion matrix and accuracy

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")

joblib.dump(model, 'models.dat')
```

```
print('Model saved: {}'.format('models.dat'))

#pedestrian detection
from imutils.object_detection import non_max_suppression
import imutils
from skimage import color
from skimage.transform import pyramid_gaussian
modelFile = 'models.dat'
inputFile = r'/Users/syha/Downloads/testimage.png'
outFile = r'/Users/syha/Downloads/testout.jpg'
image = cv2.imread(inputFile)
image = cv2.resize(image,(400,256))
size = (64,128)
step_size = (9,9)
downscale = 1.05

#List to store the detections
detections = []

#the current of scale of 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 list 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) > 0.95:
                detections.append(
                    (int(x * (downscale ** scale)), int(y * (downscale ** scale)),
model.decision_function(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 detections])
sc = [score[0] for (x, y, score, w, h) in detections]
pick = non_max_suppression(rects, probs=sc, overlapThresh=0.5)
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)

cv2.imwrite(outFile, clone)

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

Assignment 2: detect pedestrians in video clip (0.5)

Hint: using hog.detectMultiScale(frame, winStride=(8, 8))

Src 2:

```
import cv2

# Mở video để đọc
cap = cv2.VideoCapture('/Users/syha/Downloads/walk.mp4')

# Lấy chiều rộng, chiều cao và số khung hình mỗi giây (fps) của video
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)

# Xác định codec và tạo đối tượng VideoWriter
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('/Users/syha/Downloads/walkuu.mp4', fourcc, fps, (width,
height))

# Tải bộ phân loại người đi bộ HOG
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

while True:
    # Đọc từng khung hình trong video
    ret, frame = cap.read()
```

```
    # Kiểm tra nếu không có khung hình nào được trả về
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break

    # Phát hiện người đi bộ trong khung hình bằng HOG
    pedestrians, weights = hog.detectMultiScale(frame, winStride=(8, 8))

    # Vẽ các hộp giới hạn xung quanh người đi bộ được phát hiện
    for (x, y, w, h) in pedestrians:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # Ghi khung hình vào tệp video đầu ra
    out.write(frame)

    # Hiển thị khung hình có người đi bộ được phát hiện
    cv2.imshow('frame', frame)

    # Đợi phím 'q' được nhấn để thoát
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Khi mọi thứ hoàn thành, giải phóng tài nguyên
cap.release()
out.release()
cv2.destroyAllWindows()
```