HOG - Tutorial 6

Automatic Image Analysis (Technische Universität Berlin)

HOG

May 19, 2021

1 Exercise 6 - Pedestrian Detection using Histogram of Oriented Gradients

1.1 Topics:

- Data Preparation
- Feature extraction: Histogram of Oriented Gradients
- Classification: Support Vector Machine
- Object Detection: Sliding Window + Non Maximum Supression

1.2 Sources & Material

- Datasets:
  - https://www.cis.upenn.edu/~jshi/ped_html/
  - http://pascal.inrialpes.fr/data/human/
- Dataset preprocessed: https://github.com/RashadGarayev/PersonDetection
- Paper: Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR05
  - https://hal.inria.fr/inria-00548512/document/

1.3 Further Reading

- https://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/

2 1. Import Libraries

- OpenCV
- scikit-image
- scikit-learn

[1]:
```
import cv2 as cv
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import exposure
import os
from sklearn import svm
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪average_precision_score, f1_score, recall_score, precision_score
```

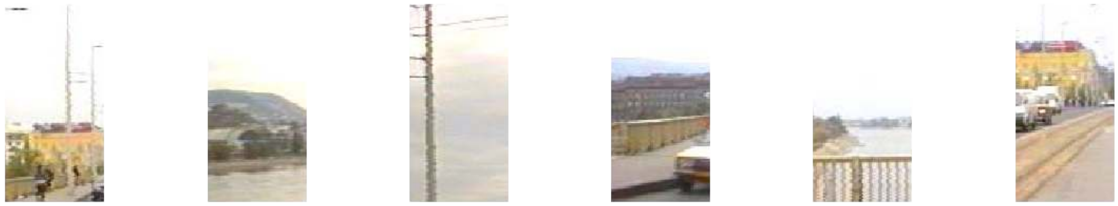
```python
import utils
%config Completer.use_jedi = False
```

3 2. Prepare Data

1. Create a dataset by extracting patches 64×128
   - positive samples that contain pedestrians
   - negative samples that contain background objects
2. Split data into training and test data

[2]:
```python
def ls(path):
    files =  os.listdir(path)
    return [os.path.join(path,x) for x in files]

positive_files = ls("dataset/positive/")
negative_files = ls("dataset/negative/")

split_pos = int(0.7*len(positive_files))
split_neg = int(0.7*len(negative_files))

train_positive, test_positive =  positive_files[:
 ↪split_pos],positive_files[split_pos:]
train_negative, test_negative =  negative_files[:
 ↪split_neg],negative_files[split_neg:]

train_positive = [cv.imread(img)[:,:,::-1] for img in train_positive]
train_negative = [cv.imread(img)[:,:,::-1] for img in train_negative]

test_positive = [cv.imread(img)[:,:,::-1] for img in test_positive]
test_negative = [cv.imread(img)[:,:,::-1] for img in test_negative]
```

[3]:
```python
utils.showImages(train_positive[:6], cols=6)
utils.showImages(train_negative[:6], cols=6)
```

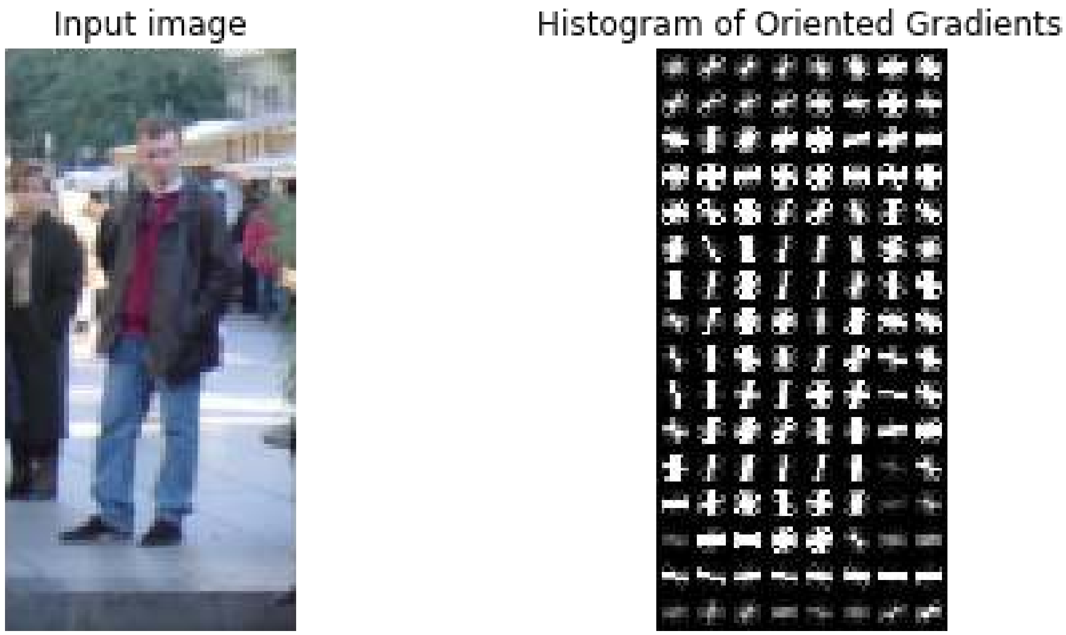


4 3. Feature Extraction

1. Apply Histogram of Oriented Gradients on all data
2. Create Training Labels

4.1 3.1 HOG

[4]:
```
img = train_positive[0]
fd, hog_image = hog(img, orientations=8, pixels_per_cell=(8, 8),
                    cells_per_block=(2, 2), visualize=True, multichannel=True)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

ax1.axis('off')
ax1.imshow(img, cmap=plt.cm.gray)
ax1.set_title('Input image')

# Rescale histogram for better display
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

ax2.axis('off')
ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```



[5]:
```python
def feature_extraction(img, ppc = 8, cpb = 2):
    return hog(img, orientations=8, pixels_per_cell=(ppc, ppc),␣
↪cells_per_block=(cpb, cpb), visualize=False, multichannel=True)

# Training Samples
train_positive_hog = [feature_extraction(img) for img in train_positive]
train_negative_hog = [feature_extraction(img) for img in train_negative]
train_positive_hog = np.array(train_positive_hog)
train_negative_hog = np.array(train_negative_hog)

# Testing Samples
test_positive_hog = [feature_extraction(img) for img in test_positive]
test_negative_hog = [feature_extraction(img) for img in test_negative]
test_positive_hog = np.array(test_positive_hog)
test_negative_hog = np.array(test_negative_hog)
```

5 3.2 Labels

[6]:
```python
# Combine positive and negative samples
X = np.concatenate((train_positive_hog,train_negative_hog))
X_test = np.concatenate((test_positive_hog,test_negative_hog))

# Create labels
Y = np.zeros(len(X))
Y[:len(train_positive_hog)] = 0
```

```
Y[len(train_positive_hog):] = 1


Y_test = np.zeros(len(X_test))
Y_test[:len(test_positive_hog)] = 0
Y_test[len(test_positive_hog):] = 1
```

6 4. Classifier

1. Initialize Model
2. Train Model
3. Evaluate Model

7 4.1 & 4.2

[7]:
```
svc = svm.LinearSVC(C=0.6,max_iter=10000)
svc.fit(X,Y)
```

[7]:
```
LinearSVC(C=0.6, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=10000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

8 4.3 Evaluate Model

[9]:
```
Y_pred = svc.predict(X_test)
```

9 Confusion Matrix

$$C = \begin{pmatrix} \text{True Negatives} & \text{False Positives} \\ \text{False Negatives} & \text{True Positives} \end{pmatrix}$$

[11]:
```
confusion_matrix(Y_test,Y_pred)
```

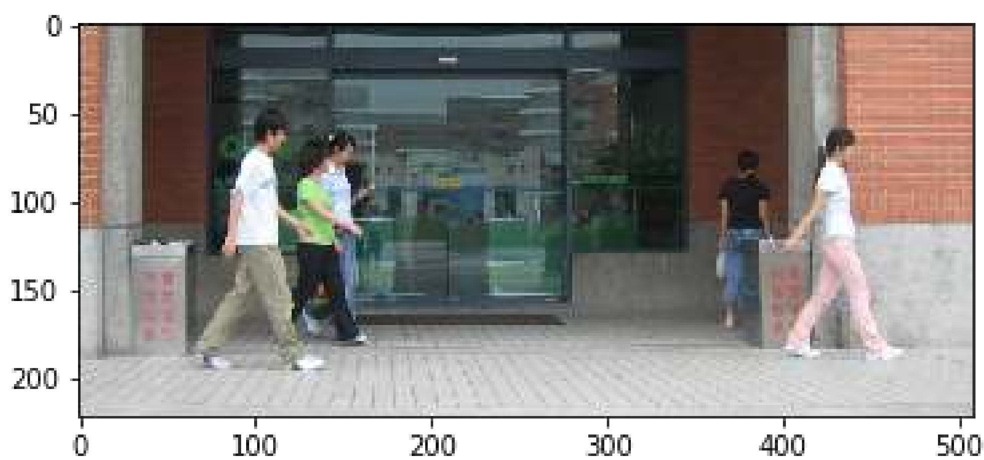
[11]:
```
array([[ 650,   75],
       [  58, 1178]], dtype=int64)
```

[12]:
```
print("AP:", average_precision_score(Y_test,Y_pred))
print("Accuracy:", accuracy_score(Y_test,Y_pred))
print("Recall:", recall_score(Y_test,Y_pred))
print("Precision:", precision_score(Y_test,Y_pred))
print("F1:", f1_score(Y_test,Y_pred))
```

```
AP: 0.9256036283199668
Accuracy: 0.9321774604793472
Recall: 0.9530744336569579
Precision: 0.9401436552274541
F1: 0.9465648854961832
```

10 5. How to use the classifier on real images

[13]:
```python
#img = cv.imread("persons/persons/person_387.bmp")[:,:,::-1]
#img = cv.resize(img, (int(0.25*img.shape[1]),int(0.25*img.shape[0])))
# 089
# 92
img = cv.imread("PennFudanPed/PNGImages/FudanPed00036.png")[:,:,::-1]
scale = 0.5
img = cv.resize(img, (int(scale*img.shape[1]),int(scale*img.shape[0])))
plt.imshow(img)
plt.show()
```



[14]:
```python
height, width = 128, 64
import utils
def detect(img, svc, step = 10, height = 128, width = 64):

    # Iterate over image and apply classifier
    result = np.zeros(img.shape[:2])-1
    for i in range(0,img.shape[0]-height, step):
        for j in range(0,img.shape[1]-width, step):
            result[i,j] = svc.predict(feature_extraction(img[i:i+height,j:
↪j+width]).reshape(1, -1))[0]

    y,x = np.nonzero(result==0)
```

```
    bounding_boxes = np.stack([x,y,x+width,y+height],1)
    return result, bounding_boxes

result, bounding_boxes = detect(img, svc)
bounding_boxes = utils.non_max_suppression_fast(bounding_boxes, 0.5)
```

[15]:
```
res = img.copy()
for i in range(len(bounding_boxes)):
    res = cv.rectangle(res.astype(np.
 ↪float32),(bounding_boxes[i,0],bounding_boxes[i,1]),(bounding_boxes[i,2],bounding_boxes[i,3]
 ↪astype(np.uint8)
plt.imshow(res)
```

[15]: <matplotlib.image.AxesImage at 0x27db19fa1c8>