Lab Instructions - session 14

Object Detection
Aref Azizpour

HoG-SVM

In the previous lab session, we used the SVM with HoG features to classify objects. In this session, we want to detect the location of the objects as well. First, we need to extract the HoG feature from our dataset (both positive and negative samples). Then we train an SVM on these features. Once our model is trained properly, we can perform object detection on new images using a sliding window approach (like the `detectMultiScale` method in OpenCV) and use its output to draw bounding boxes around the detected objects. Your job is to change the parameters of the classifier and detectMultiScale to achieve the best result possible. Below you can see a sample code for training and testing the detector. It is up to you to choose the number of data samples needed for training, but the more the better. You can find more samples here. Notice that the data might need preprocessing.

File: **HoGSVM.py**

```
import cv2
import numpy as np
import glob
from sklearn import svm
from sklearn.externals import joblib
import itertools
import imutils
from imutils.object_detection import non_max_suppression
import os

hog = cv2.HOGDescriptor('hog.xml')
train_data = []
train_labels = []
X = []

# reading positive samples
fnamesPos = glob.glob('pos/*.png')

for fname in fnamesPos:
    I1 = cv2.imread(fname)
    I1 = I1[3:-3, 3:-3, :]
    feature = hog.compute(I1)
    train_data.append(feature)
    train_labels.append(1)
```

```
# reading negative samples
fnamesNeg = glob.glob('neg/*.png')

for fname in fnamesNeg:
    I1 = cv2.imread(fname)
    # creating some random negative samples from images which don't
contain any pedestrians.
    samples1 = np.random.randint(0, I1.shape[1] - 64, 10)
    samples2 = np.random.randint(0, I1.shape[0] - 128, 10)
    samples = zip(samples1, samples2)
    for sample in samples:
        I2 = I1[sample[1]:sample[1] + 128, sample[0]:sample[0] + 64, :]
        feature = hog.compute(I2)
        train_data.append(feature)
        train_labels.append(0)

X = np.asarray(train_data, dtype=np.float64)
X = np.reshape(X, (X.shape[0], X.shape[1]))
train_labels = np.asarray(train_labels)

# training the SVM classifier
file_name = 'saved_svm.sav'

classifier = svm.SVC(kernel='sigmoid', C=1, tol=1e-4, coef0=1,
gamma='scale', max_iter=-1)

if not os.path.isfile(file_name):
    print('training the SVM')
    classifier.fit(X, train_labels)
    joblib.dump(classifier, file_name)
else:
    classifier = joblib.load(file_name)
    print('saved classifier loaded')

# putting the final support vector params in correct order for feeding to
our HoGDescriptor
supportVectors = []
supportVectors.append(np.dot(classifier.dual_coef_,
classifier.support_vectors_)[0])
supportVectors.append([classifier.intercept_])
supportVectors = list(itertools.chain(*supportVectors))
hog.setSVMDetector(np.array(supportVectors, dtype=np.float64))

# testing
scale = 1.05
padding = (4, 4)
winStride = (8, 8)

fnamesTest = glob.glob('test/*.png')
for fname in fnamesTest:
    I = cv2.imread(fname)
    I = imutils.resize(I, width=min(400, I.shape[1]))
    (rects, weights) = hog.detectMultiScale(I, winStride=winStride,
padding=padding, scale=scale,
                                            useMeanshiftGrouping=True)
    rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
    rects = non_max_suppression(rects, probs=None, overlapThresh=0.6)
```

```
    for (xA, yA, xB, yB) in rects:
        cv2.rectangle(I, (xA, yA), (xB, yB), (0, 255, 0), 2)
    cv2.imshow("detected pedestrians", I)
    cv2.waitKey(0)
```

- What are the positive and negative samples? What are they used for? How are they generated?
- Explain what does detectMultiScale do and its parameters.
- Explain SVM.SVC parameters.

Cascade Detection

Cascade detection is a way of building a fast detector by the early rejecting of the image areas not containing an object. In each step the classifier gets more accurate and more complex and slower as a result. In this approach, we discard the vast majority of the windows not containing objects very quickly and put more time and effort on windows that “may” contain the object we are looking for. In this session, we only use pre-trained cascades opencv has provided. We use an LBP-cascade for face detection and a haar-cascade for smile detection. You can find more cascade classifiers here. Also, you can make your own classifier by using the methods explained here and here.

File: **task.py**

```
import numpy as np
import cv2

#https://github.com/opencv/opencv/blob/master/data/lbpcascades/lbpcascade_
frontalface_improved.xml
face_detector = cv2.CascadeClassifier('face.xml')
#https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascad
e_smile.xml
smile_detector = cv2.CascadeClassifier('smile.xml')

cap = cv2.VideoCapture(0)

while 1:
    ret, I = cap.read()

    #complete the code here in order to detect your face and your smile
    cv2.imshow('I',I)
    key = cv2.waitKey(60)
    if key == ord('q'):
        break
```

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

References and further reading

[sklearn.svm.SVC]
[INRIA Person Dataset]
[Rapid Object Detection using a Boosted Cascade of Simple Features (paper)]
[hog.detectMultiScale parameters explained]
[cv2.HOGDescriptor]
[cv2.CascadeClassifier]