SIT789 – Robotics, Computer Vision and Speech Processing

Pass Task 4.2: Object detection

Objectives

Object detection is a fundamental task in Computer Vision. In this lab, the objectives include:

- Practising local image features and classifiers for object detection from still images and videos
- Working with two case studies: face detection and pedestrian detection

---

Tasks

1. Face detection

1.1. Face detection from still images

In this section, we practise with **rectangular features** and **Cascade detector** built upon sliding window object detection and boosted cascade in the task of human face detection from still images from the paper.

> `P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. CVPR, 2001`

You first download the face image database provided in FaceImages.zip in OnTrack, unzip this file to a folder named FaceImages into your working directory. The images in this folder are collected from FDDB: Face Detection Data Set and Benchmark.

You are also provided with a pre-trained cascade detector whose details are provided in in haarcascade_frontalface_default.xml in OnTrack. You also need to download this xml file into your working directory. You can find other pre-trained detectors (detecting other object types, e.g., profile faces, full human body, left/right eyes, license plates, etc.) at https://github.com/opencv/opencv/tree/master/data/haarcascades.

The following detect_face method receives input as a colour image and a cascade detector, then detects human faces from the input image, and returns a list of detected human faces (if any). Although this code is provided to you, you are encouraged to read through and understand it. You should revisit Week 3 slides and recording for more details about object detection.

```
import cv2 as cv

def detect_face(image, cascade_detector):
    #convert input image to grayscale
    image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    faces = cascade_detector.detectMultiScale(image_gray,
            scaleFactor = 1.1, #the ratio between two consecutive scales, must be > 1
            minNeighbors = 5, #minimum number of overlapping windows to be considered
            minSize = (30, 30)) #minimum size of a detection window (in pixels)

    return faces
```

We test the above detect_face method with the supplied images in FaceImages folder and the cascade detector defined in haarcascade_frontalface_default.xml as follows.

```
import time

input_image = cv.imread('FaceImages/abba.png')
cascade_detector = cv.CascadeClassifier('haarcascade_frontalface_default.xml')

start_time = time.time()
faces = detect_face(input_image, cascade_detector)
print('Face detection is performed in %s seconds ---' % (time.time() - start_time))
```

To find the number of detected faces, we perform:

```
if (faces is not None):
    print('Found ', len(faces), ' faces')
else:
    print('There is no face found!')
```

To visualise the detection results, we perform:

```
from matplotlib import pyplot as plt

output_image = input_image.copy()
for (x, y, w, h) in faces: #(x, y) are the coordinates of the topleft corner,
                           #w and h are the width and height of the bounding box
    cv.rectangle(output_image, (x, y), (x + w, y + h), (0, 255, 0), 2)

plt.imshow(output_image[:,:,::-1]) # RGB-> BGR
```

You can experiment the cascade face detector with more challenging images from the WIDER face dataset. If you are interested in training the cascade detector with your own data, you can find the instructions and training procedure from here.

Your task now is to understand the effect of the parameters used the detectMultiScale method. In particular,

1. Vary scaleFactor in the range [1.1, 1.5, 2.0] and observe the corresponding detection results and processing time on the supplied face image dataset.
2. Vary minNeighbors in the range [0, 5, 50] and observe the corresponding detection results and processing time on the supplied face image dataset.

3. Vary minSize in the range [(10, 10), (30, 30), (100, 100)] and observe the corresponding detection results and processing time on the supplied face image dataset.
4. Find the best parameters based on your own observation. Apply these parameters to the cascade_detector to detect human faces in images img_1014.jpg and img_1123.jpg. Are all the faces detected? If not, what are the missing faces? What could be the reason?

1.2. Face detection from video

In this section, we will experiment the cascade detector with video data captured by webcams. The following code performs face detection directly from your webcam. Note that this code cannot be used on Google Colab.

```
import cv2 as cv
import os

#initialise webcam
cam = cv.VideoCapture(0)

#initialise cascade_detector
cascade_detector = cv.CascadeClassifier('haarcascade_frontalface_default.xml')

while True:
    #read the image from the cam
    _, image = cam.read()

    #detect human faces from the current image using the cascade_detector
    faces = detect_face(image, cascade_detector)

    #display detected faces
    for x, y, w, h in faces:
        cv.rectangle(image, (x, y), (x + w, y + h), color = (0, 255, 0))

    cv.imshow('face detection demo', image)

    if cv.waitKey(1) == ord("q"):
        break

cam.release()
cv.destroyAllWindows()
cv.waitKey(1)
```

If you cannot turn off the webcam after the face detection is completed, you can try the following setups.

- Open cmd (for Windows)
- Type: `setx OPENCV_VIDEOIO_PRIORITY_MSMF 0`
- Restart computer

Your task in this section is to execute the above code and observe detection results in the following cases:

1. Move your face near and far from the webcam.
2. Rotate your face in different angles.
3. Hide parts of your face, e.g., use your hand to hide one eye, wear sunglasses, use your hand to hide the forehead, use your hand or scarf to hide the mouth.
4. Change the lighting condition in the surrounding environment.

2. Pedestrian detection

In this section, we will experiment the HOG pedestrian detector developed in the following paper,

> N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. CVPR, 2005.

This work makes use of HOG as image features and SVM as classifier.
We first need to install **imutils** which provides utilities, e.g., image resizing, non-maximum suppression, by performing the following steps:

- Open cmd (for Windows) or terminal (for Macs)
- Type `pip install imutils`. If **imutils** has been installed on your computer, you may update it using `pip install --upgrade imutils`

You also need to download the pedestrian image database provided in PedestrianImages.zip in OnTrack, unzip this file to a folder named PedestrianImages into your working directory. The images in this folder are from the INRIA person dataset made by Dalal and Triggs. You can find the full version of the INRIA person dataset and other pedestrian benchmarks at http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/. For a review on pedestrian detection, you are referred to the following paper,

> D. T. Nguyen et al. Human detection from images and videos: a survey. Pattern Recognition, vol. 51, pp. 148-175, 2016.

The following detect_pedestrian method receives input as a colour image, then detects pedestrians from the input image, and returns a list of detected pedestrians (if any). Although this code is provided to you, you are encouraged to read through and understand it.

```
import cv2 as cv
import numpy as np
import imutils
from imutils.object_detection import non_max_suppression

def nms(boxes):
    #Convert boxes from list to array as required by the non_max_suppression method
    #Each box in the array is encoded by the topleft and bottomright corners
    boxes_array = np.array([[x, y, x + w, y + h] for (x, y, w, h) in boxes])
    boxes_array = non_max_suppression(boxes_array, probs = None, overlapThresh = 0.65)

    #create a new list of boxes to store results
    boxes_list = []
    for top_x, top_y, bottom_x, bottom_y in boxes_array:
        boxes_list.append([top_x, top_y, bottom_x - top_x, bottom_y - top_y])

    return boxes_list

def detect_pedestrian(image):
    #initialise the HOG descriptor and SVM classifier
    hog = cv.HOGDescriptor()
    hog.setSVMDetector(cv.HOGDescriptor_getDefaultPeopleDetector())

    image_resized = imutils.resize(image,
                    width = min(400, image.shape[1])) #resize the input image so that
                                                      #the width is max by 400
    scale = image.shape[1] / image_resized.shape[1]
```

```
    #detect pedestrians
    (boxes, _) = hog.detectMultiScale(image_resized,
                   winStride = (4, 4), #horizontal and vertical stride
                   padding = (6, 6), #horizontal and vertical padding for each window
                   scale = 1.05) #scale factor between two consecutive scales

    #non-maximum suppression
    boxes = nms(boxes)

    #resize the bounding boxes
    for box in boxes:
        box[0] = int(box[0] * scale)
        box[1] = int(box[1] * scale)
        box[2] = int(box[2] * scale)
        box[3] = int(box[3] * scale)

    return boxes
```

We test the above detect_pedestrian method with the images in PedestrianImages as follows.

```
import time

image = cv.imread('PedestrianImages/person_222.png')

start_time = time.time()
pedestrians = detect_pedestrian(image)
print('Pedestrian detection is performed in %s seconds ---' % (time.time()-start_time))
```

We show the detection results using the following code

```
from matplotlib import pyplot as plt

if (pedestrians is not None):
    print('Found ', len(pedestrians), ' pedestrians')

    for (x, y, w, h) in pedestrians:
        cv.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 10)
    plt.imshow(image[:,:,::-1]) # RGB-> BGR
else:
    print('There is no pedestrian found!')
```

Your task now is to test the above code with images in PedestrianImages and explore the following aspects.

1. Comment the line boxes = nms(boxes) and observe detection results.
2. Vary padding in the range {(0, 0), (6, 6)} and observe the corresponding detection results and processing time on the supplied pedestrian dataset.
3. Vary winStride in the range {(4, 4), (8, 8), (12, 12)} and observe the corresponding detection results and processing time on the supplied pedestrian dataset.
4. Vary scale in the range {1.05, 1.2} and observe the corresponding detection results and processing time on the supplied pedestrian dataset.

Submission instructions

1. Perform tasks required in Section 1 and 2.
2. Complete the provided answer sheet and submit the answer sheet (.pdf) to OnTrack.