DATA.ML.300 Computer Vision Exercise 3

Miska Romppainen
H274426

January 2021

1 Task: HOG descriptors for people detection

Here is my solution to task 1. HOG descriptors for people detection. Idea of the task was to implement a simple people detector using Histogram of Oriented Gradients as descriptor to detect humans from a GIF. I used Python as my choice of programming language. Image from the original GIF can be seen in the figure 1.

Figure 1: Original image from the running GIF

Histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection.[**1**] Basic idea of HOG is to calculate the occurrence of different gradient orientations in a specific space of an image. By calculating these gradient orientations a histogram of the orientations can be created, this is histogram of orientated gradients or so called HOG.

Using function *HOGDescriptor* from *cv2*, you can input different parameter values for the HOG descriptor. Values such as scale, win stride, padding and hit threshold. Scale controls the factor in which the image is resized at each layer of the Gaussian image pyramid and therefore also influencing the number

of levels in the image pyramid. Win stride controls the "step-size" of the sliding window, or in other words how big steps the sliding window takes in each iteration. Padding indicates the number of pixels in both the x and y direction in which the sliding window ROI is "padded" prior to HOG feature extraction. Hit threshold is the threshold for the distance between features and SVM classifying plane, this parameter is used to remove false positives.
Using different parameter values outputs different detected outputs, but some of the detected outputs might be computationally heavier than others. A output of a relatively "fast" process with its parameters can be seen in figure 2 and a output of a slow process in figure 3.

CODE SNIPPET FROM: *hog_detector.py*

```
#HOG Parameter values for fast process
scale = 1.05
winStride = (4, 4)
padding = (16, 16)
hitThreshold = 0.0
.
.
detections, _ = hog.detectMultiScale(img,
                                     winStride=winStride,
                                     padding=padding,
                                     scale=scale,
                                     hitThreshold=hitThreshold)

# Draw bounding boxes around the detected objects
for (x,y,w,h) in detections:
    #cv2.rectangle(image, start_point, end_point, color, thickness)
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
.
.
.
# Exit cleanup
cv2.destroyAllWindows()
```

As we can see, in the background of the fast image (figure 2) the other humans are not detected. By tuning and trying other parameter values the output does not change, even though the parameters would be tuned in a way that its computationally more heavy or more "accurate". Output of the slower image can be seen in the figure 3.

CODE SNIPPET FROM: *hog_detector.py*

```
#HOG Parameter values for slow process
scale = 1.01
winStride = (1, 1)
```

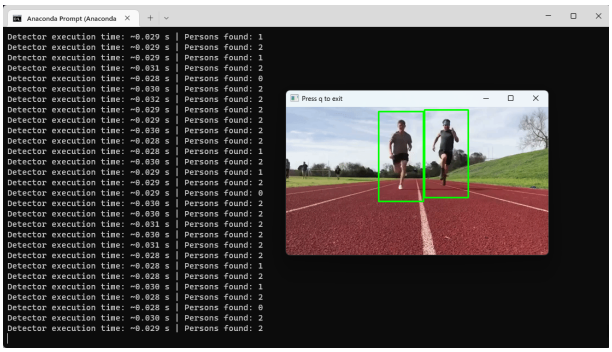


Figure 2: "Fast" detection using HOG with parameters: scale=1.05, winStride=(4,4), padding=(16,16), hitThreshold=0.0

```
padding = (0, 0)
hitThreshold = 0.0
```

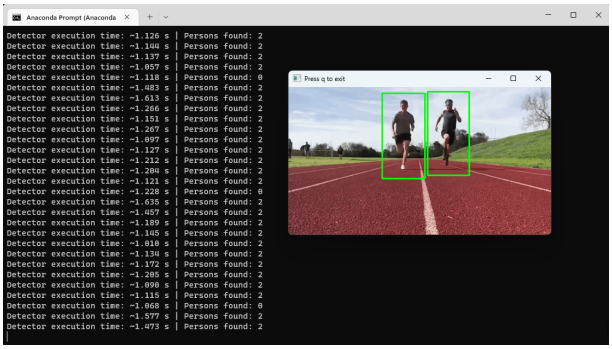


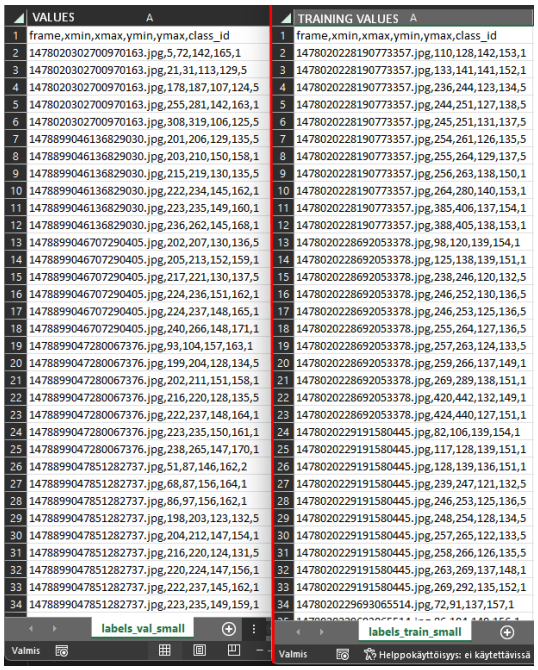
Figure 3: "Slow" detection using HOG with parameters: scale = 1.01, winStride = (1, 1), padding = (0, 0), hitThreshold = 0.0

2 Task: Basics of SSD object detector

Here is my solution to task 2. Basics of SSD object detector. Idea of the task was to introduce and dive into CNN based object detector, Single Shot Multi-Box Detector (SSD) and answer questions in the exercise PDF. These questions were the following:
1. We are using (a small portion of) the Udacity road traffic dataset as an example, where the target objects are vehicles and traffic lights. The dataset can be found in the datasets directory and the target values can be found in the .csv files. What is the form the targets are presented in? What is the difference between training and validation datasets in a general sense?
2. Next we'll take a look at the general architecture of the model. The keras_ssd7.py file implements a smaller version of the SSD detector. Open keras_ssd7.py under the models-directory, and locate the build_model function. Try to find where the first convolutional part (before the convolutional predictor layers) of the network is defined. How many convolutional "blocks" are there, and what kind of layers is each block build from?
3. SSD has it's own loss function, defined in chapter 2.2 in the original publication. What are the two attributes this loss function observes? How are these defined (short explanation without any formulas is sufficient)? The publication can be found in the exercise folder or here.

Question 1: *What is the form the targets are presented in? What is the difference between training and validation datasets in a general sense?*

In the figure 4 we can values saved in the training and validation datasets of the udacity traffic dataset. In the figure 6 we can see all the images saved inside *udacity_driving_datasets*-folder. The form the values are saved in is: image from which the data is fetched from, the coordinates for x_{min} and x_{max}, coordinates for y_{min} and y_{max} and class identifier for the detected object inside the coordinates.



	VALUES
1	frame,xmin,xmax,ymin,ymax,class_id
2	1478020302700970163.jpg,5,72,142,165,1
3	1478020302700970163.jpg,21,31,113,129,5
4	1478020302700970163.jpg,178,187,107,124,5
5	1478020302700970163.jpg,255,281,142,163,1
6	1478020302700970163.jpg,308,319,106,125,5
7	1478899046136829030.jpg,201,206,129,135,5
8	1478899046136829030.jpg,203,210,150,158,1
9	1478899046136829030.jpg,215,219,130,135,5
10	1478899046136829030.jpg,222,234,145,162,1
11	1478899046136829030.jpg,223,235,149,160,1
12	1478899046136829030.jpg,236,262,145,168,1
13	1478899046707290405.jpg,202,207,130,136,5
14	1478899046707290405.jpg,205,213,152,159,1
15	1478899046707290405.jpg,217,221,130,137,5
16	1478899046707290405.jpg,224,236,151,162,1
17	1478899046707290405.jpg,224,237,148,165,1
18	1478899046707290405.jpg,240,266,148,171,1
19	1478899047280067376.jpg,93,104,157,163,1
20	1478899047280067376.jpg,199,204,128,134,5
21	1478899047280067376.jpg,202,211,151,158,1
22	1478899047280067376.jpg,216,220,128,135,5
23	1478899047280067376.jpg,222,237,148,164,1
24	1478899047280067376.jpg,223,235,150,161,1
25	1478899047280067376.jpg,238,265,147,170,1
26	1478899047851282737.jpg,51,87,146,162,2
27	1478899047851282737.jpg,68,87,156,164,1
28	1478899047851282737.jpg,86,97,156,162,1
29	1478899047851282737.jpg,198,203,123,132,5
30	1478899047851282737.jpg,204,212,147,154,1
31	1478899047851282737.jpg,216,220,124,131,5
32	1478899047851282737.jpg,220,224,147,156,1
33	1478899047851282737.jpg,222,237,145,162,1
34	1478899047851282737.jpg,223,235,149,159,1

labels_val_small

Valmis

	TRAINING VALUES
1	frame,xmin,xmax,ymin,ymax,class_id
2	1478020228190773357.jpg,110,128,142,153,1
3	1478020228190773357.jpg,133,141,141,152,1
4	1478020228190773357.jpg,236,244,123,134,5
5	1478020228190773357.jpg,244,251,127,138,5
6	1478020228190773357.jpg,245,251,131,137,5
7	1478020228190773357.jpg,254,261,126,135,5
8	1478020228190773357.jpg,255,264,129,137,5
9	1478020228190773357.jpg,256,263,138,150,1
10	1478020228190773357.jpg,264,280,140,153,1
11	1478020228190773357.jpg,385,406,137,154,1
12	1478020228190773357.jpg,388,405,138,153,1
13	1478020228692053378.jpg,98,120,139,154,1
14	1478020228692053378.jpg,125,138,139,151,1
15	1478020228692053378.jpg,238,246,120,132,5
16	1478020228692053378.jpg,246,252,130,136,5
17	1478020228692053378.jpg,246,253,125,136,5
18	1478020228692053378.jpg,255,264,127,136,5
19	1478020228692053378.jpg,257,263,124,133,5
20	1478020228692053378.jpg,259,266,137,149,1
21	1478020228692053378.jpg,269,289,138,151,1
22	1478020228692053378.jpg,420,442,132,149,1
23	1478020228692053378.jpg,424,440,127,151,1
24	1478020229191580445.jpg,82,106,139,154,1
25	1478020229191580445.jpg,117,128,139,151,1
26	1478020229191580445.jpg,128,139,136,151,1
27	1478020229191580445.jpg,239,247,121,132,5
28	1478020229191580445.jpg,246,253,125,136,5
29	1478020229191580445.jpg,248,254,128,134,5
30	1478020229191580445.jpg,257,265,122,133,5
31	1478020229191580445.jpg,258,266,126,135,5
32	1478020229191580445.jpg,263,269,137,148,1
33	1478020229191580445.jpg,269,292,135,152,1
34	1478020229693065514.jpg,72,91,137,157,1

labels_train_small

Valmis Helppokäyttöisyys: ei käytettävissä

Figure 4: Image of labels_val_small.csv (left) and labels_train_small.csv (right)

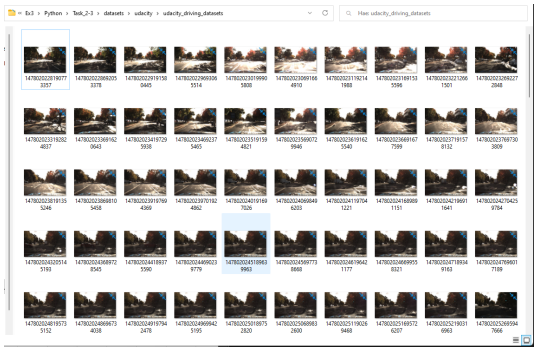


Figure 5: Screenshot of the folder of all the images

The general difference the datasets in training and evaluation datasets are different. For example in the training dataset all of the images start with the number series "*147802022*". In the evaluation and training datasets only share one image as cross-reference the image "*1478020302700970163.jpg*", otherwise the datasets are completely different.

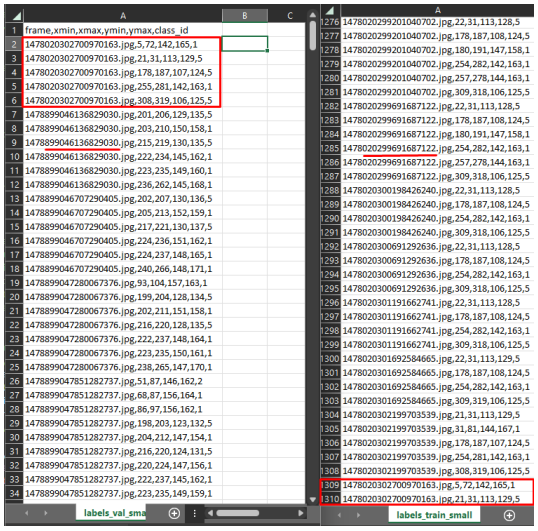


Figure 6: Screenshot of the differences between evaluation and training datasets

Question 2: *Try to find where the first convolutional part (before the convolutional predictor layers) of the network is defined. How many convolutional "blocks" are there, and what kind of layers is each block build from?*

The network building happens before adding the convolutional predictor layers on top of the base network that was defined. In the default case four predictor layers are used. Keras_ssd7 implementation has 7 convolutional layers and 4 convolutional predictor layers that take their input from layers 4, 5, 6, and 7, respectively.
Convolutional predictor layers on top of conv layers 4, 5, 6, and 7, for these two predictor layers we build on top of each of these layers: One for class prediction (classification), one for box coordinate prediction (localization) after which we predict confidence for each box (class predictor) and the box coordinates for each box (box predictor).

Question 3: *SSD has it's own loss function, defined in chapter 2.2 in the original publication. What are the two attributes this loss function observes? How are these defined (short explanation without any formulas is sufficient)?*

According to the SSD original publication; "The overall objective loss function is a weighted sum of the localization loss (loc) and the confidence loss (conf)" and the weight term α (which is set to 1 by cross validation)

The localization loss (loc) is a Smooth L1 loss between the predicted box (l) and the ground truth box (g) parameters. Smooth L1 loss is a combination of the mean squared error (MSE) loss and the absolute error (L1) loss. The function is smooth near the target value and becomes less sensitive to outliers as the difference between the predicted value and the target value increases.

The confidence loss (conf) is the softmax loss over multiple classes confidences (c). Softmax loss measures the difference between the predicted probability distribution and the true distribution for a given input. The softmax function is applied to the predicted values to ensure that they are in the form of probabilities, which sum up to 1.

3 Task: SSD300 for real-time object detection

Here is my solution to task 3. SSD300 for real-time object detection. Idea of the task was to use a larger version of SSD with pretrained weights to implement real-time object detection for webcam feed.
Some of the detected objects from a can be seen in the figures 7, 8 and 9

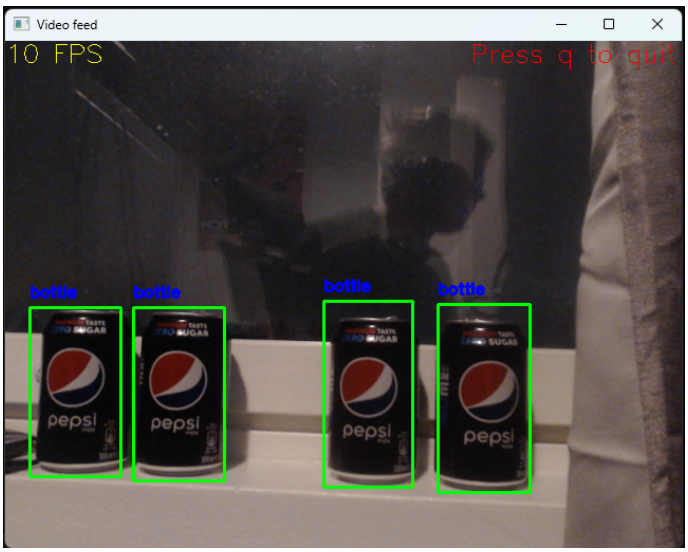


Figure 7: Screenshot of bottles taken from output

In order to create the real-time object detection, *ssd_300* Keras model was created with weighted paths. These paths were available from the exercise PDF. In the *ssd300_webcam.py* instructions for the task were given. The implemented code can be seen in the code snippet below.

In the code, different predetermined class identifiers for detected objects were declared, such identifiers were for example 'background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car' etc. After the implementation of the real-time object detection, these identified objects could be detected from a live feed from a web-camera. These detection outputs can be seen in the figures 7 and 9. The accuracy of detections is not 100% accurate, sometimes items can be falsely detected. An example of falsely detected object can be seen in 9. However the detection for items in the predetermined objects is pretty accurate and the code is fun to play with using real-time detection.

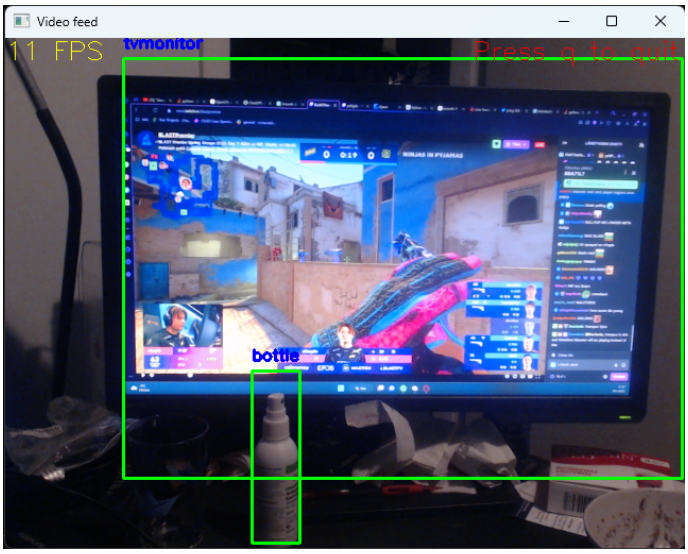


Figure 8: Detected TV (PC) monitor and an bottle

CODE SNIPPET FROM: *ssd300_webcam.py*

```
# camFlipped = cv2.flip(cam,1) # Mirror the frame
# (this can be omitted in case using a video clip)

# Capture frame-by-frame
ret, img = cam.read()

resized = imresize(img, (img_width, img_height), preserve_range=True)

resized = np.array([resized])

# Get the original frame dimensions
original_height, original_width  = img.shape[:2]

# Calculate the ratio of original and reshaped width and height
width_ratio  =  original_width / 300
height_ratio =  original_height / 300

# [class_id, confidence, xmin, ymin, xmax, ymax] = model.predict(resized)
for prediction in model.predict(resized):
    for row in prediction:
        if row[1] > confidence_threshold: # omit 0 confidence values
        # Transform the bounding box coordinates
            xmin = int(row[2] * width_ratio)
```

```
ymin = int(row[3] * height_ratio)
xmax = int(row[4] * width_ratio)
ymax = int(row[5] * height_ratio)

# Draw the bounding box on the original frame
# cv2.rectangle(image, start_point, end_point, color, thickness)
cv2.rectangle(img, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

# Put the class label on the bounding box
cv2.putText(img, c[int(row[0])], (xmin, ymin-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
```

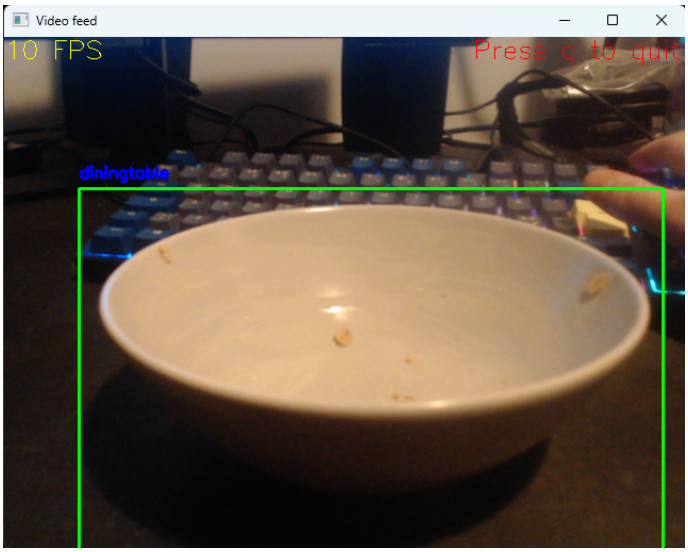


Figure 9: Some items not in the predetermined list of detectable items outcomes funny

References

[1] Wikipedia, Histogram of oriented gradients (2023), https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients, Last accessed 25 January 2023