Assignment 1 Report

Task 1

task 1a)

The Intersection over Union (IoU) is an evaluation metric to measure the accuracy of a prediction in object detection. We find it by calculating the area (sum of pixels) of the interesection between two overlapping bounding boxes divided by the union of the same. In Figure 1 we see how the green boxes illustrate the original object detection area, i.e. the ground-truth bounding boxes, while the red ones illustrate the models predicted bounding box. We may then calulate the Intersection over the union by diving the area A over B. Hence, we see that the IoU is limited between 0 and 1 and the higher the number means higher accuracy.

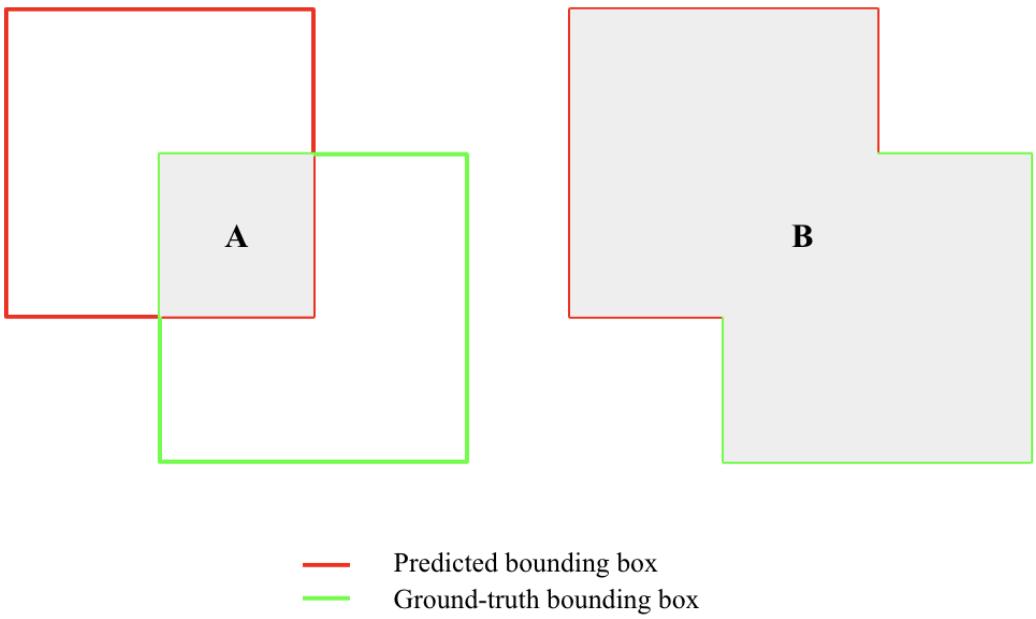


Figure 1: *Illustration of the intersection (left) and union (right) between the ground-truth and predicted bounding boxes*

task 1b)

A true positive (TP) is a correct prediction of a class, while a false positive (FP) is an incorrect prediction of a class. Equations for precision and recall are given by

$$precision = \frac{TP}{TP + FP}$$
$$recall = \frac{TP}{TP + FN},$$

respectively. FN represents a false negative, that is, a prediction that something is not an object when it in fact was.

task 1c)

The average precision for a class is given by

$$AP = \int_0^1 p(r)dr.$$

We approximate this by calculating the average precision for each class using recall levels at $0.0, 0.1, 0.2, \ldots, 0.9, 1.0$ (See figure below for code plotting corresponding curves). The average precision for each class thus becomes

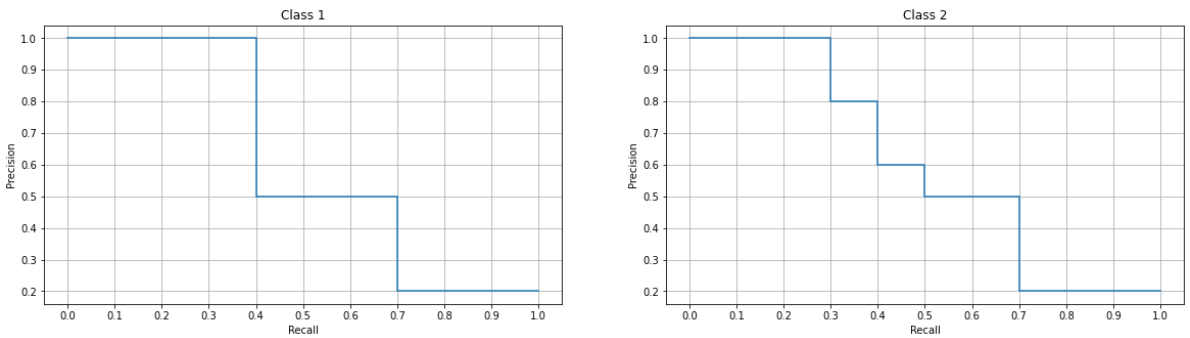
$$AP_{C_1} = \frac{(5 \times 1 + 3 \times 0.5 + 3 \times 0.2)}{11} = 0.6455,$$
$$AP_{C_2} = \frac{(4 \times 1 + 1 \times 0.8 + 1 \times 0.6 + 2 \times 0.5 + 3 \times 0.2)}{11} = 0.6364.$$

To obtain the mean average precision (mAP), we then take the mean over all classes C to obtain

$$mAP = \frac{1}{|C|} \sum_{c \in C} AP_c = \frac{0.6455 + 0.636}{2} = 0.641.$$

In [28]:

```
import matplotlib.pyplot as plt
import numpy as np

# Precision and recall curve for class 1:
Precision1 = [1.0, 1.0, 1.0, 1.0, 0.5, 0.20]
Recall1 = [0.0, 0.05, 0.1, 0.4, 0.7, 1.0]

# Precision and recall curve for class 2:
Precision2 = [1.0, 1.0, 0.80, 0.60, 0.5, 0.20]
Recall2 = [0.0, 0.3, 0.4, 0.5, 0.7, 1.0]

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
x_axis = np.linspace(0.0, 1.0, 11)

ax1.step(Recall1, Precision1)
ax1.set_xlabel('Recall')
ax1.set_xticks(x_axis)
ax1.set_ylabel('Precision')
ax1.set_title('Class 1')
ax1.grid(True)

ax2.step(Recall2, Precision2)
ax2.set_xlabel('Recall')
ax2.set_xticks(x_axis)
ax2.set_ylabel('Precision')
ax2.set_title('Class 2')
ax2.grid(True)

plt.show()
```



Task 2

Task 2f)

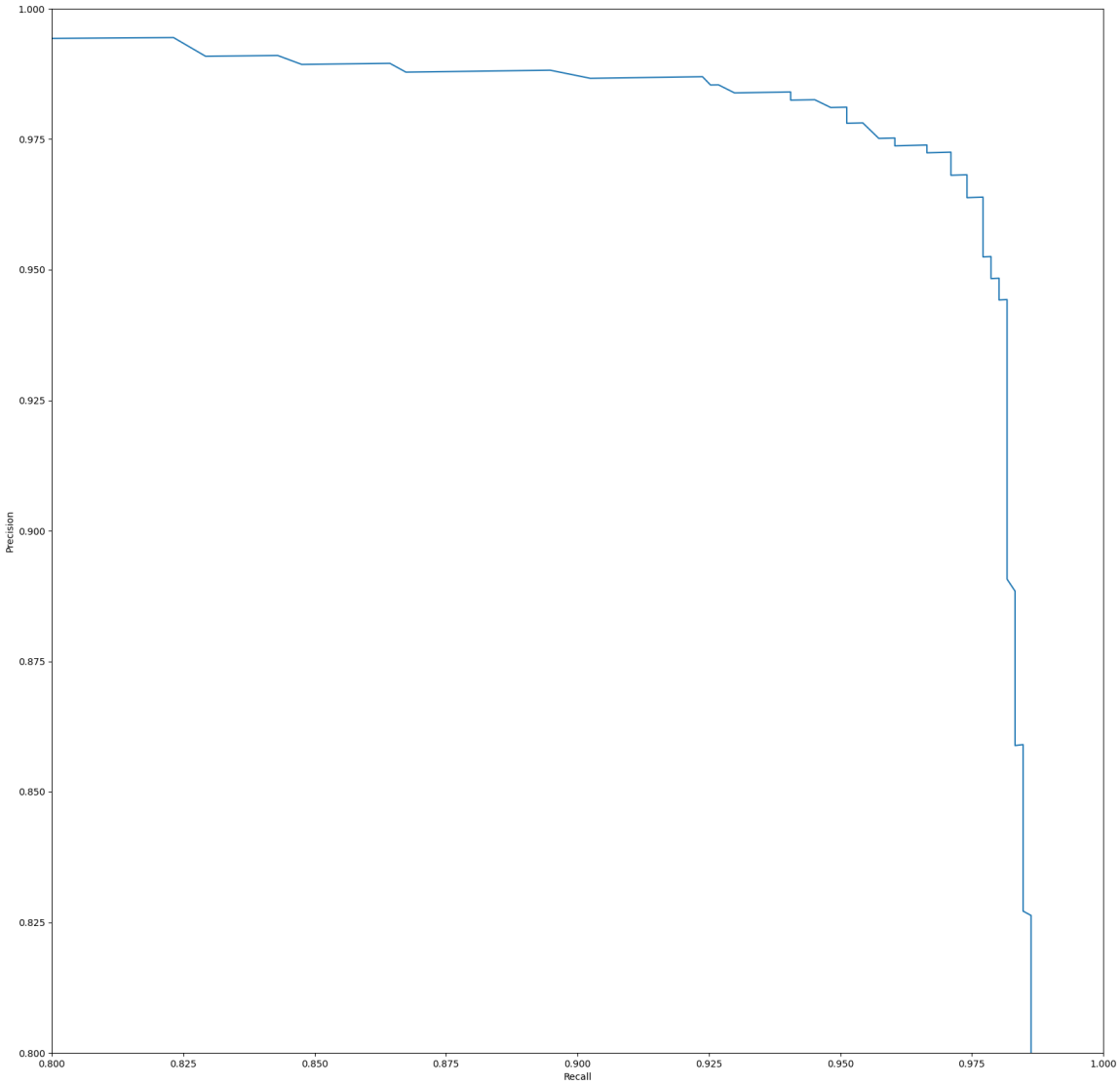


Figure 2: *Final precision-recall curve*

Task 3

Task 3a)

The filtering operation used in SSD is called non-maximum suppression (nms).

Task 3b)

The statement is false. The opposite is true, that the predictions from deepers layers detect larger objects.

Task 3c)

The reason for using different bounding box aspect ratios at the same spatial location is to obtain faster and more stable convergence. This happens since the different classes can have different shapes, and aspect ratios detect different objects and shapes.

Task 3d)

The main difference between SSD and YOLOv1/v2 is that the latter used a single scale feature map for detection, while the first uses multi-scale feature maps.

Task 3e)

For this feature map we would have a total of $38 \times 38 \times 6 = 8,664$ anchor boxes.

Task 3f)

There is a total of $6 \times (38 \times 38 + 19 \times 19 + 10 \times 10 + 5 \times 5 + 3 \times 3 + 1 \times 1) = 11,640$ anchor boxes for the entire network.

Task 4

Task 4b)

Figure 3 shows the total loss over ~11k iterations while the final mean average precision (mAP@0.5 (mailto:mAP@0.5)) was 0.7536.

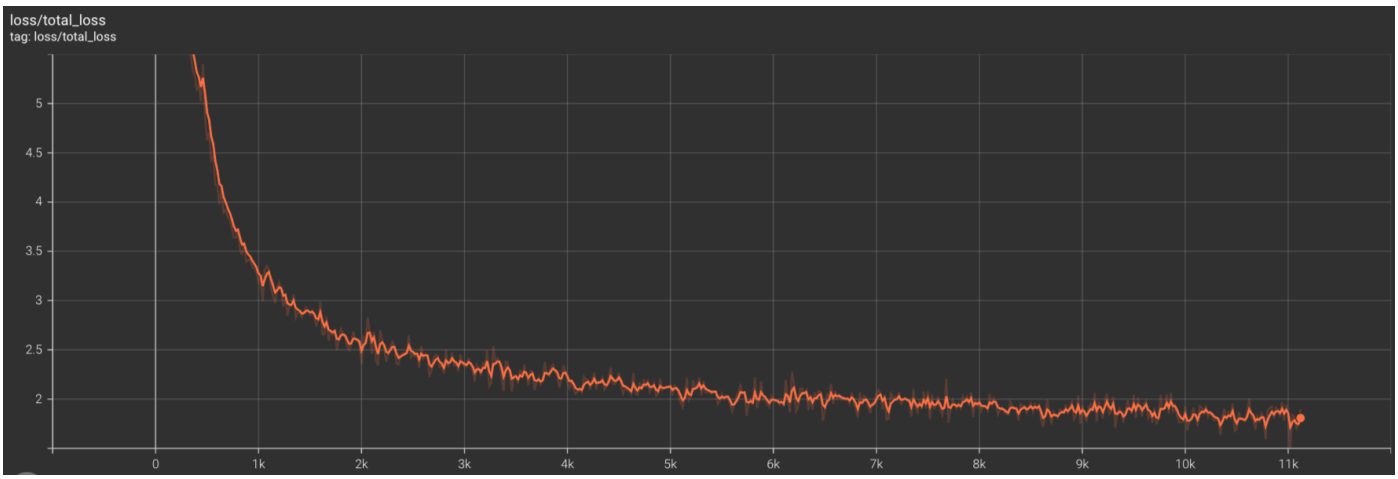


Figure 3: *loss/total loss over 11k iterations.*

Task 4c)

The code for the improved model can be found in `Task4c.py`.

Using the lessons learnt from the last assignment, the best improvement was obtained using batch normalization after the convolutional layers. However, the actual improvement from this was not paricularly visible with the SSD model.

In an attempt to further increase the performance different optimizers were tested. The Adam optimizer proved to worsen the performance, while the Adagrad optimizer resulted in a slight improvement.

After changing back to the SGD optimizer, the next step was to double the number of filters from 32 to 64 as that proved to increase the performance of previous models. While this increased the number of parameters and training time sufficiently, it also drastically increased the performance. This achieved a mAP of 82.26% after 8,736 iterations as shown in Figure 4.

Final mAP was ... and the results are shown in Figure 4.

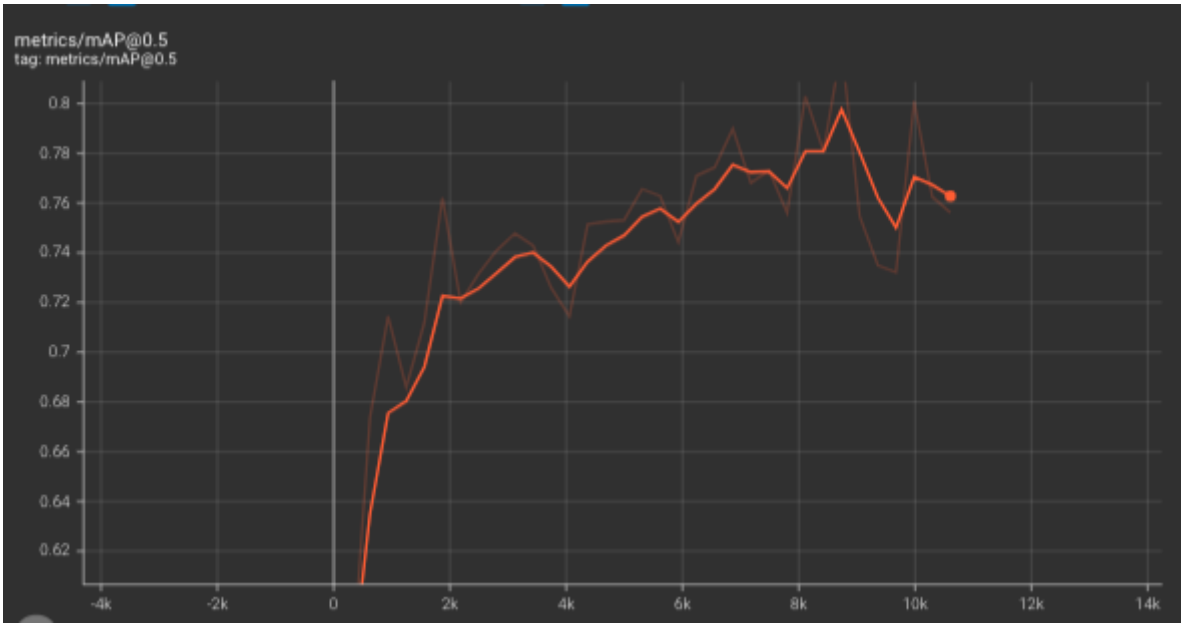


Figure 4: *Best modelling after improvements.*

Task 4d)

Given a 300x300 image and 25 center points and shapes of 6 anchor boxes we obtain the result in Figure 5. The pixel valeues for 25 centers are $[32 + 64n, 32 + 64n]$ where $n \in 0, 1, 2, \ldots, 4$ and the shapes of siz anchor boxes are [162, 162], [186, 186], [229, 115], [115, 229], [94, 281], [281, 94].

Running the notebooks/visualize priors results in the results shown in Figure 6.

Figure 5: .

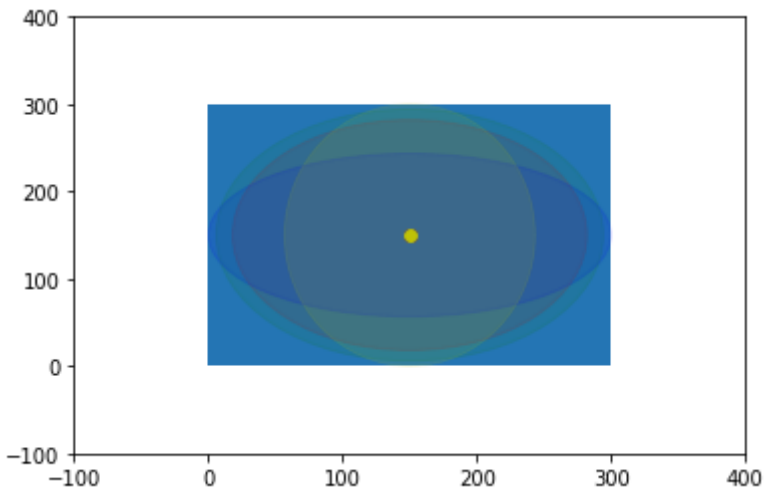


Figure 5: .

Task 4e)

The model was able to detect all large numbers, but struggled with some of the smaller ones. This can be seen in Figure 7 and Figure 8.

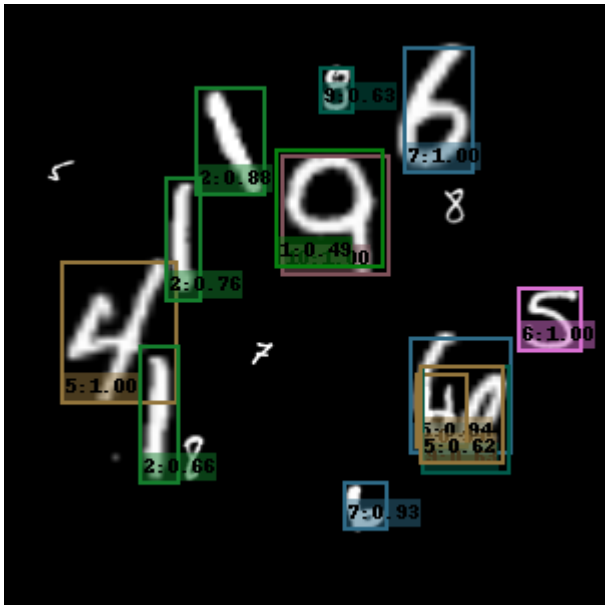


Figure 7: *Best model ran on some digits. Most can be identified, except for the smallest ones.*

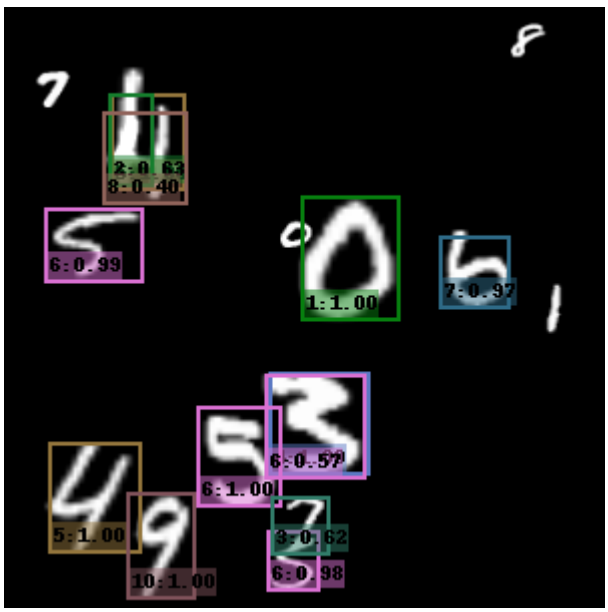


Figure 8: *Best model ran on some digits. Most can be identified, except for the smallest ones.*

Task 4f)

Final map@0.5 (mailto:map@0.5) was 0.519.

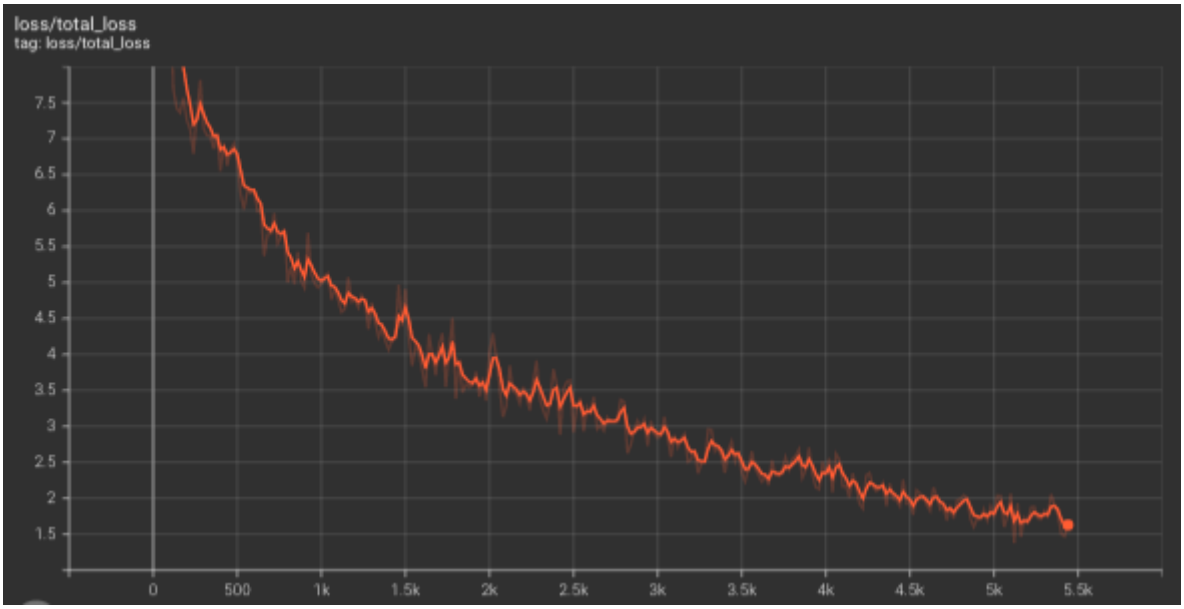


Figure 7: *Total loss after 5k iterations of training voc_vgg.py model.*