Assignment 04
TDT4265 - Datasyn og Dyplæring

Elias Mohammed Elfarri
Christopher Michael Vibe

Mars, 2021

Contents

1 Task 1 - Object Detection Metrics

1.a

The intersection over union, or IoU for short, is the intersection of two sets A, B (purple area) divided by the union of sets A, B.

$$IoU = \frac{(A \cap B)}{(A \cup B)} \tag{1.1}$$

In terms of bounding boxes the concept it the same, but the circles then represent the areas of the bounding boxes and the object in question. This metric measures how much the two overlap, and thus indicate how well the bounding box encapsulates the object.

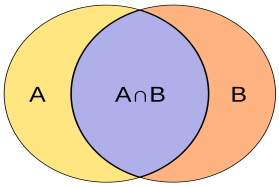


Figure 1: Intersection over union, graphically presented.

1.b

Some abbreviations:

$$\begin{aligned} True\_Positives &= TP \\ False\_Positives &= FP \\ False\_Negatives &= FN \\ True\_Negatives &= TN \end{aligned} \tag{1.2}$$

Intuitive definition of **precision**: Of the cases predicted positive, how many where correctly classified?

$$precision = \frac{TP}{TP + FP} \tag{1.3}$$

Intuitive definition of **recall**: Of all existing positive cases, how many where correctly classified positive?

$$Recall = \frac{TP}{TP + FN} \tag{1.4}$$

True positive - a predicted outcome where a model correctly predicts the positive class.

False positive - also known as type 1 error in statistical terms is a predicted outcome where a model wrongly predicts the positive class when it should be a negative.

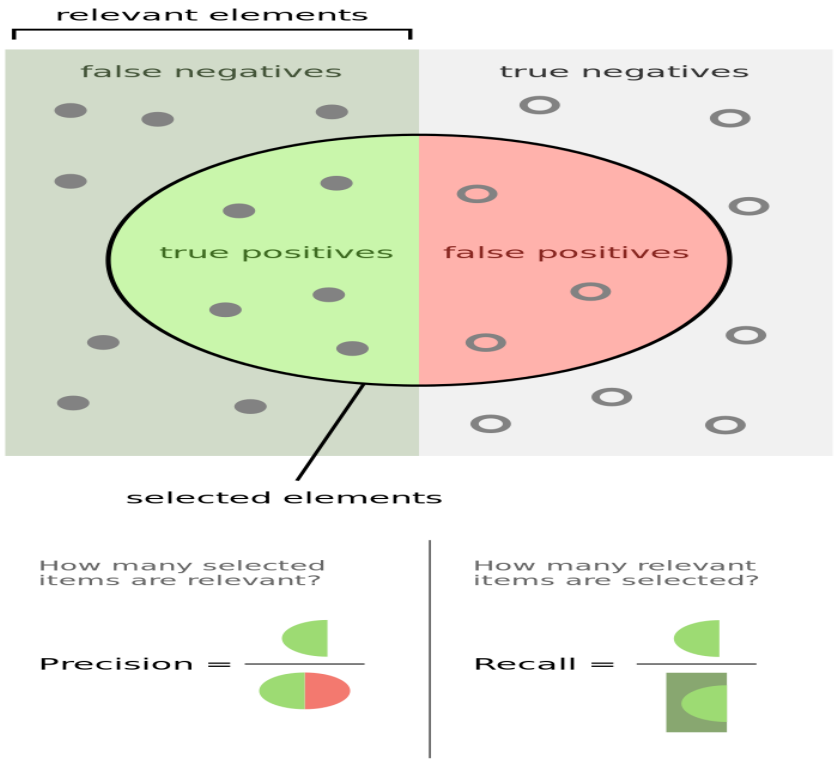


Figure 2: Visualization of task 1b. Image Source: https://en.wikipedia.org/wiki/Precision_and_recall

1.c

The average precision is calculated by first interpolating the measured data-points for precision and recall. The max values from the right are graphically "smeared" left for a conservative calculation which also creates a characteristic stair shape. The next step is to calculate the mean for the interpolated points corresponding to the 11 x-coordinates from 0 to 1 with a stepsize of .1 Finally the two class means are averaged; the average mean precision is obtained. The equation for an average precision of a class using the 11 point method is as follows:

$$\mathbf{AP} = \frac{1}{11} \sum_{r \in 0.0, 0.1, \ldots, 1.0} \mathbf{max}_{\tilde{r} \geq r} p(\tilde{r})$$

Where if there are different precision values for the same recall value in the range (0.0,0.1, ... ,1.0) then only the max precision value is picked out for said recall value. Giving the following calculation:

$$\mathbf{AP_1} = \frac{1}{11}(1 \cdot 5 + 0.5 \cdot 3 + 0.2 \cdot) = 0.6454$$

$$\mathbf{AP_2} = \frac{1}{11}(1 \cdot 4 + 0.8 + 0.6 + 0.5 \cdot 2 + 0.2 \cdot 3) = 0.6363$$

Giving a mAP:

$$\mathbf{mAP} = \frac{\mathbf{AP_1} + \mathbf{AP_2}}{2} = 0.64085$$

Furthermore the graph below illustrates the 11 recall values with their corresponding precision points for class 1 and class 2.

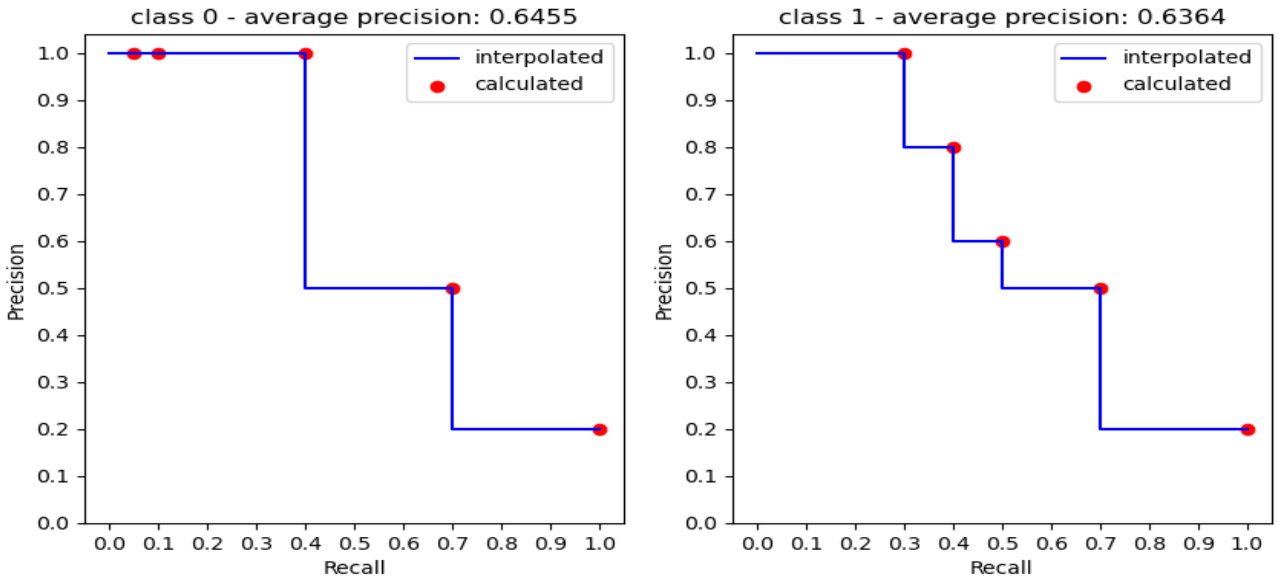


Figure 3: Average precision, visualised

2 Task 2 - Implementing Mean Average Precision

2.a

Coding task implemented in "task2.py".

2.b

Coding task implemented in "task2.py".

2.c

Coding task implemented in "task2.py".

2.d

Coding task implemented in "task2.py".

2.e

Coding task implemented in "task2.py".

2.f

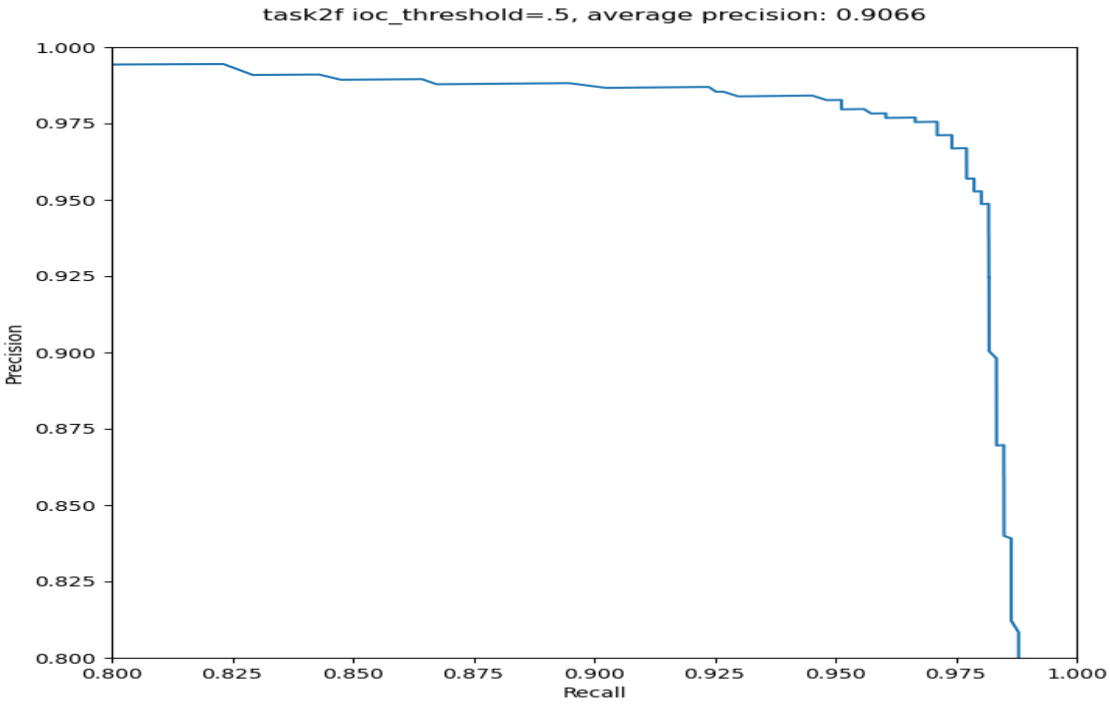


Figure 4: Precision-recall curve, with ioc=0.5, 500 confidence values for the predicting boxes are sampled to approximate the curve.

3 Task 3 - SSD Theory

3.a

The filtering operation is called **non-max suppression**. This is just a fancy name for taking the max confidence prediction and ignoring the rest.

3.b

False, from the SSD blog post by Jonathan Hui: "In SSD, small objects can only be detected in higher resolution layers (leftmost layers). But those layers contain low-level features, like edges or color patches, that are less informative for classification."

3.c

It is to reduce the cumbersome job of finding boxes that fit each type of object perfectly to match its unique shape or aspect ratio. A single square is usually not optimal, so we generalize by using a selection of bounding boxes to cover the most typical shapes. From the SSD blog post by Jonathan Hui: "To keep the complexity low, the default boxes are pre-selected manually and carefully to cover a wide spectrum of real-life objects."

3.d

The main difference between YOLO and SSD is that the classification in the SSD architecture is connected to CNN layers over several depths, catering to classification of various sized objects. Also that there are not fully connected layers in SSD, using further CNN layers instead for classification. From the SSD research paper under *related work*: "if we use the whole topmost feature map and add a fully connected layer for predictions instead of our convolutional predictors, and do not explicitly consider multiple aspect ratios, we can approximately reproduce YOLO." The result is that SSD becomes a faster network more suitable for real-time evaluation with high accuracy.

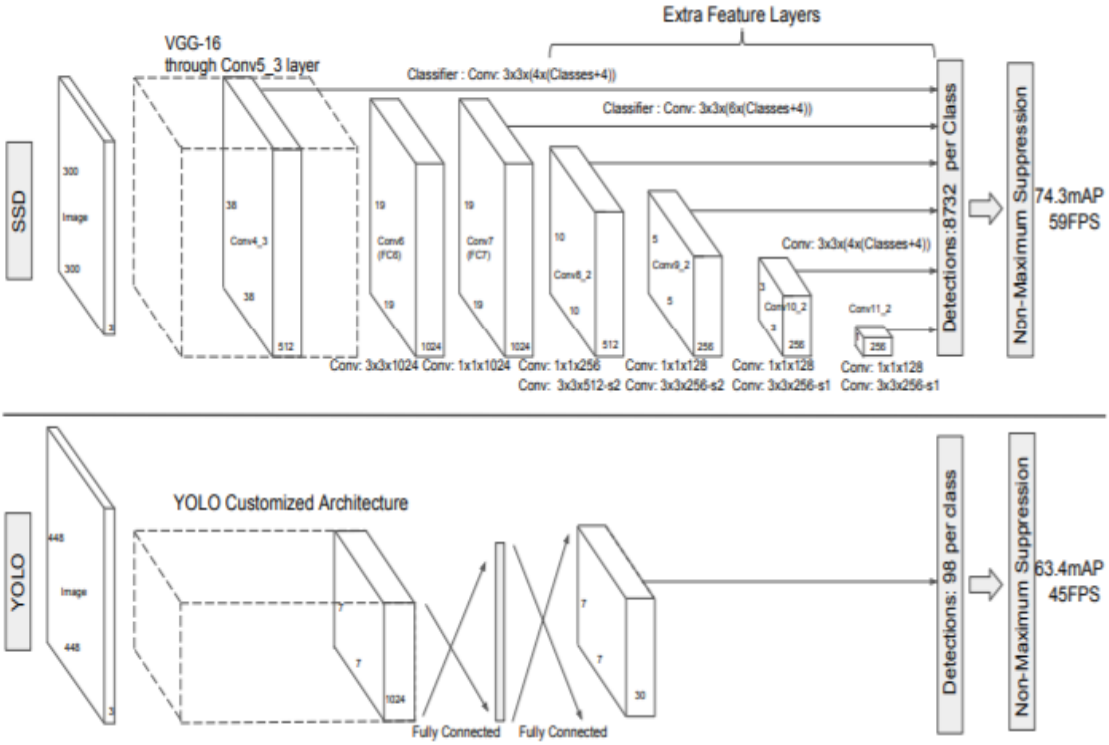


Figure 5: SSD vs YOLO architectures.

3.e

Each of the 6 anchors with different aspect ratios has each pixel on a feature map as an anchor point. Naturally only a few anchor points in a feature map will be valid, and certainly only for some of the anchors. How ever counting all the bounding boxes will achieve 38x38x6 bounding boxes for the feature map which is the same as a total of 8664 bounding boxes or 1444 of each of the 6 bounding box types.

$$38\text{x}38\text{x}6 = 1444\text{x}6 = \underline{\underline{8664}}$$

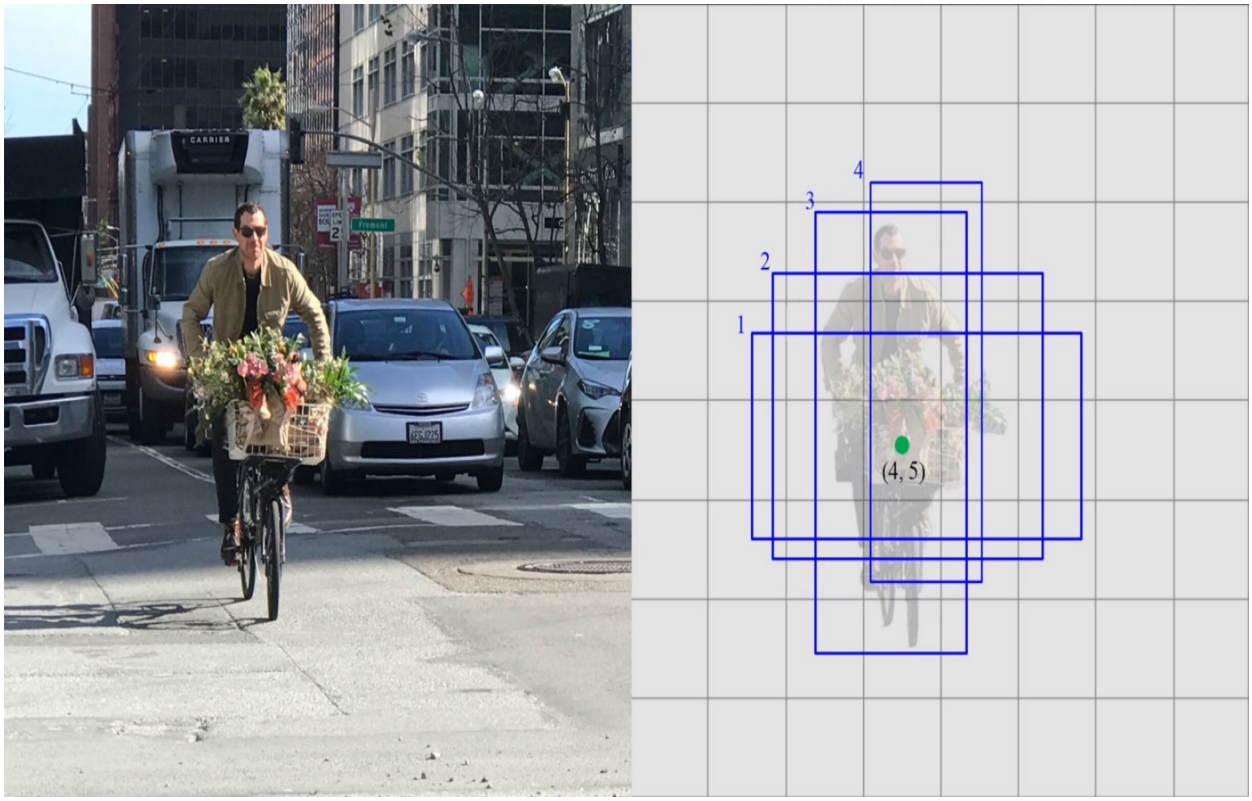


Figure 6: Illustration from Jonathan Hui's SSD blog, in the picture each grid predicts which of its 4 anchor boxes best fit the object at hand. Note the grid is actually 38x38 but was simplified for visualization to 8x8, but effectively there are 38x38x4 total bounding boxes for the feature map of this figure.

3.f

The same logic applies to 3.f as in 3.e where the total bounding boxes becomes each pixel of the feature map resolution times the different types of bounding boxes. In other words **H**x**W**x**BoundingBoxAspectRatios**. Calculations for this subtask result in:

$$38\text{x}38\text{x}6 + 19\text{x}19\text{x}6 + 10\text{x}10\text{x}6 + 5\text{x}5\text{x}6 + 3\text{x}3\text{x}6 + 1\text{x}1\text{x}6$$

$$= 8664 + 2166 + 600 + 150 + 54 + 6 = \underline{\underline{11640}}$$

4 Task 4 - Implementing Single Shot Detector

4.a

Coding task implemented in "../SSD/ssd/modeling/backbone/basic4a.py".

4.b

The final mAP of the SSD for 6000 iterations of the MNIST Object detection dataset is 75.74%.

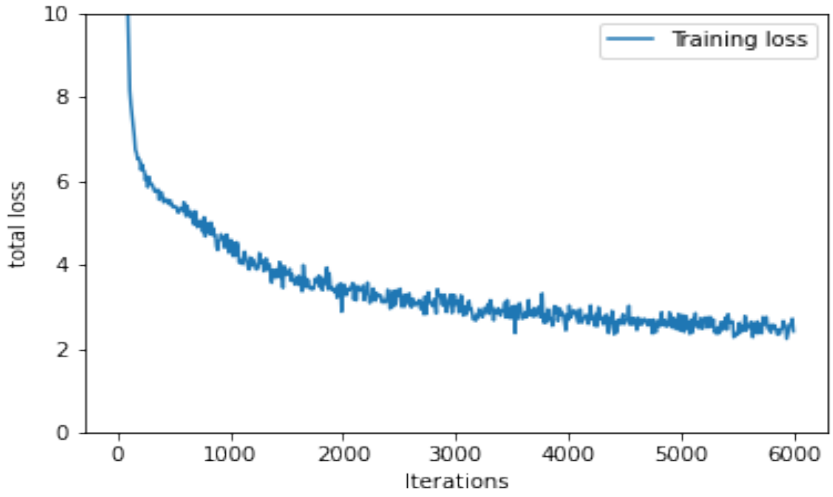


Figure 7: Training loss over 6000 iterations on the MNIST Object detection dataset.

4.c

With our improved model we achieve an mAP of 85.42% after 1K iterations. Model found under "../SSD/ssd/modeling/backbone/optimal.py".

Methods that Improved the model:

- **Batchnorm2D before each conv2D of the base model from 4a.**
- **AdamW Optimizer with LR: 5e-4.**
- **Jaccard filter threshold increased to 0.55.**
- **Doubling of all the filters at all stages of the architecture.**
- **Changing MIN SIZES parameter for the first output layer from 30 to 20.**

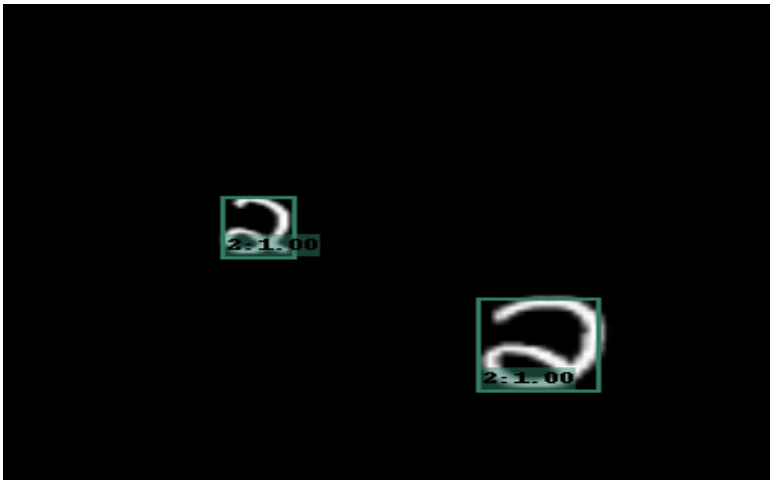
4.d

With the same improved model we achieve an mAP of 90.95% within 10K iterations.

4.e

There were four cases we noticed in particular where our model struggled with classification of the mnist numbers:

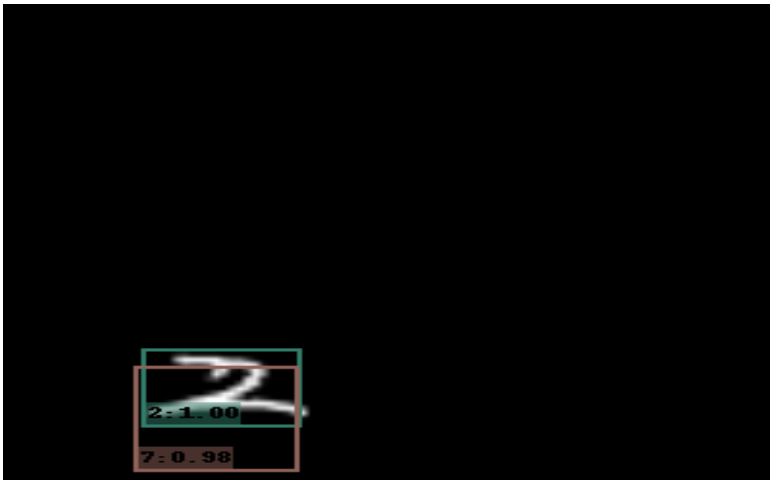
- Overlapping numbers
- Rotated numbers
- small numbers
- Non-continuous numbers

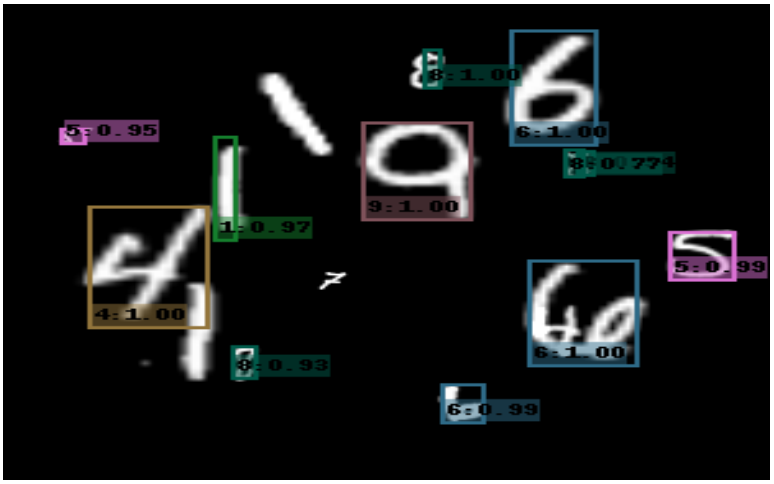


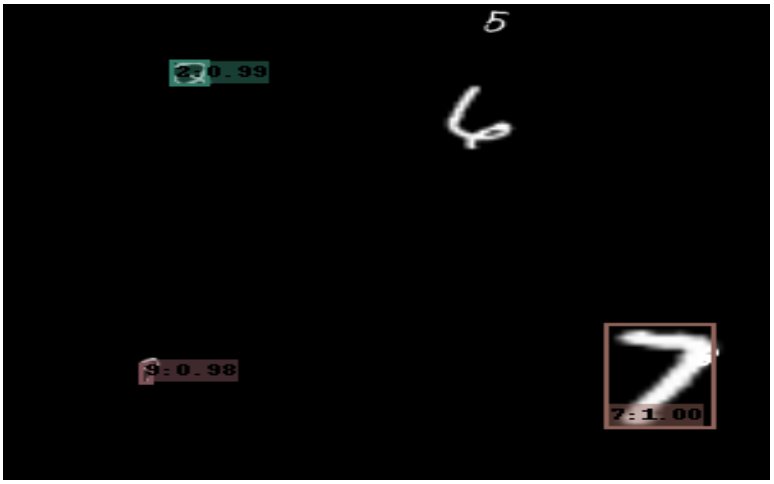

7: 0.98
5: 1.00
6: 1.00
5: 0.79
7: 1.00
3: 0.98


5: 1.00
3: 1.00
1: 0.92

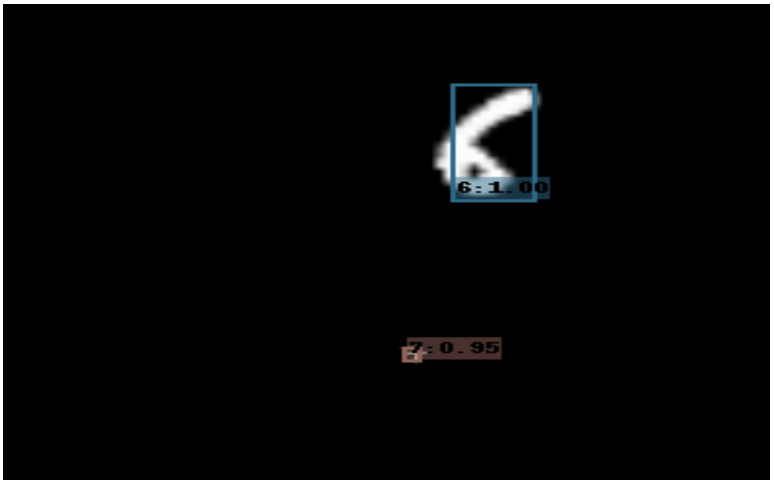

4: 0.96
2: 0.96
2: 1.00
7: 0.99
1: 0.87
1: 0.82
1: 0.94
0.97
3: 0.92
6: 1.00
8: 1.00
4: 1.00
1: 0.98
7: 0.78
1.00

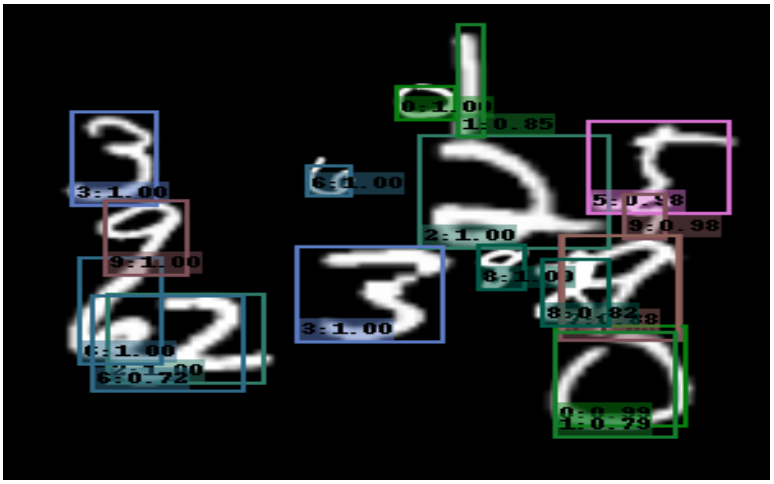

2: 1.00
7: 0.98

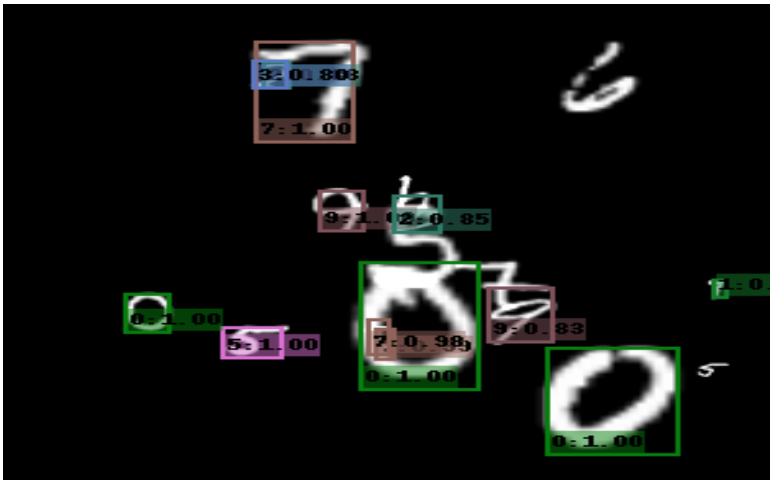

8: 1.00
6: 1.00
5: 0.95
9: 1.00
1: 0.97
4: 1.00
6: 1.00
8: 0.93
6: 0.99

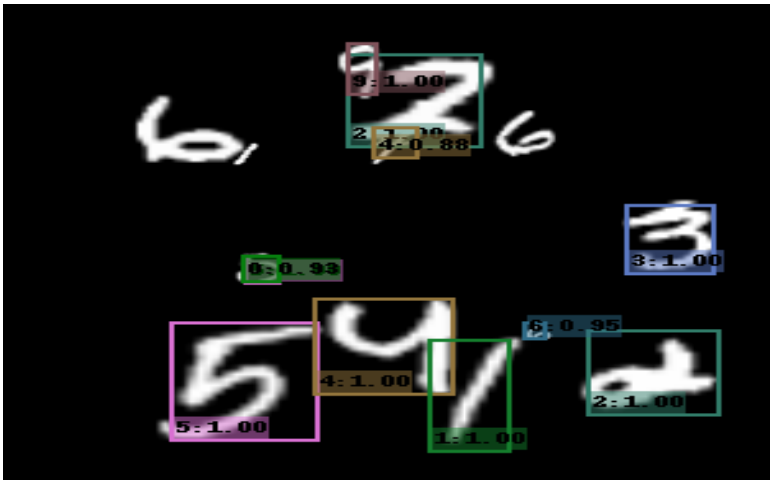

9: 0.98
7: 1.00

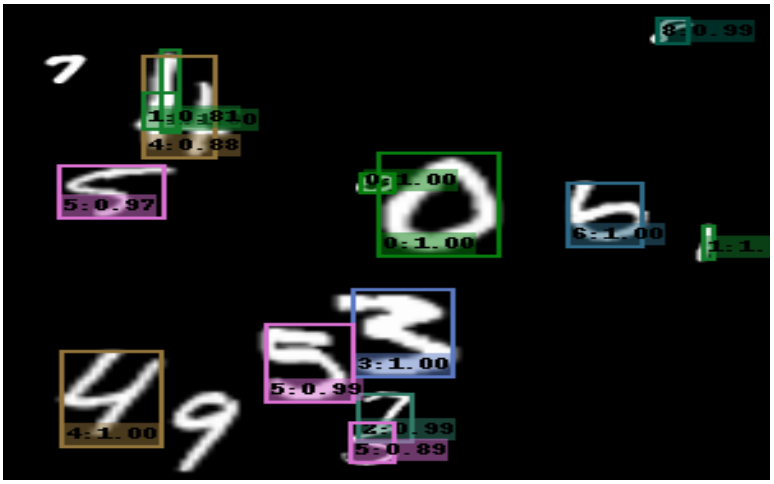

6:0.75
7:0.92
3:1.00
6:0.95
9:1.00
2:1.00
0:1.00
6:0.99
2:1.00
1.00


6:1.00
7:0.95


0:1.00
1:0.85
3:1.00
6:1.00
5:0.98
9:0.98
2:1.00
9:1.00
8:1.00
8:0.82
3:1.00
0:1.00
6:0.72
1:0.79


3: 0.803
7: 1.00
2: 0.85
0: 1.00
5: 1.00
7: 0.98
9: 0.83
0: 1.00
0: 1.00


9: 1.00
4: 0.88
3: 1.00
0: 0.93
6: 0.95
4: 1.00
2: 1.00
5: 1.00
1: 1.00


8: 0.99
1: 0.81
4: 0.88
5: 0.97
0: 1.00
0: 1.00
6: 1.00
3: 1.00
5: 0.99
4: 1.00
5: 0.89

4.f

The final mAP on the validation set was 23.35%, not very impressive but naturally also very low because this needs a lot of training. The default iteration is set to 100K which would take about a day to train on the TDT4265 cluster.

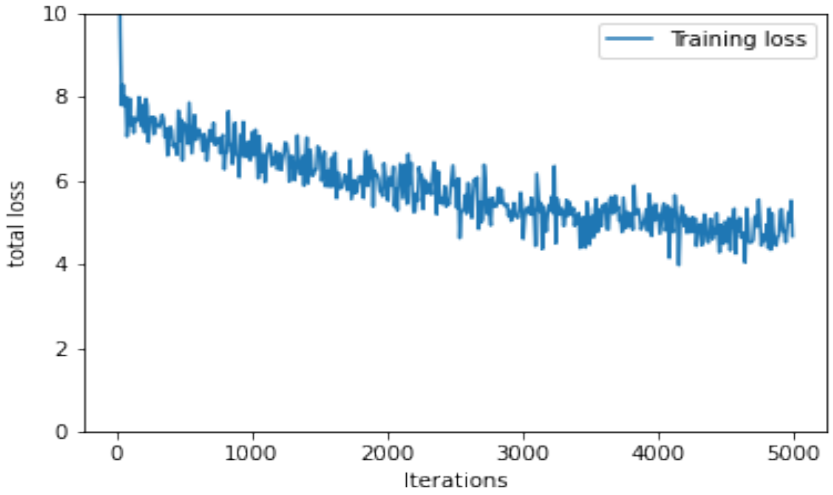


Figure 8: Training loss of the VGG16 model over 5000 iterations on the VOC dataset.

The resulting 5 classified images are below:

Figure 9: one of the 5 images classified by the trained VGG16 model.

Figure 10: one of the 5 images classified by the trained VGG16 model.

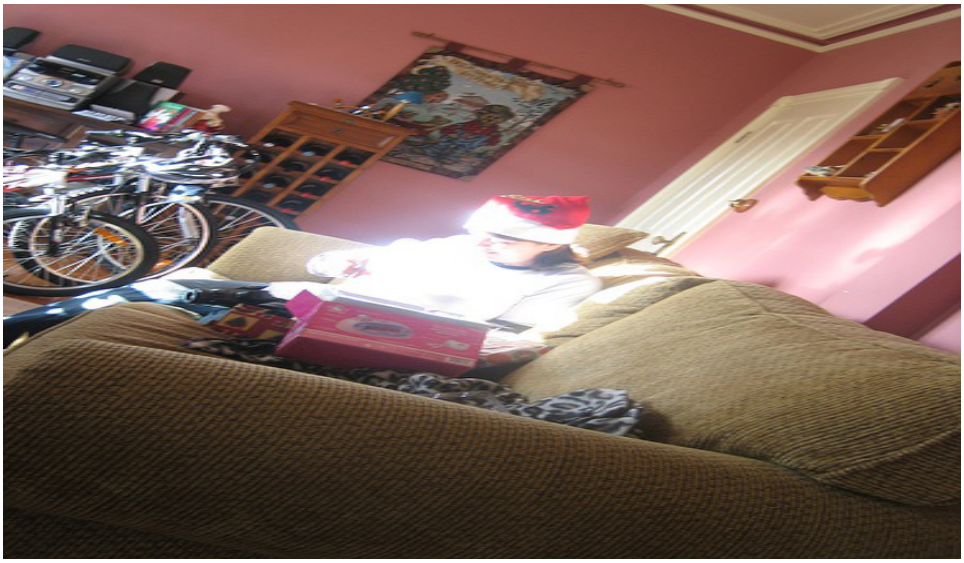


Figure 11: one of the 5 images classified by the trained VGG16 model.

Figure 12: one of the 5 images classified by the trained VGG16 model.

Figure 13: one of the 5 images classified by the trained VGG16 model.