Assignment 4 - Report

TDT4265 Computer Vision and Deep Learning

Sivert Utne

April 21, 2022

Contents

Task 1

(a)

Intersection over Union or *IoU* is simply the intersection of two boxes divided by the union of the two. In other words the IoU is higher the more of the two are overlapping. This is expressed as:

$$\text{IoU}(A, B) = \frac{\text{Intersection of } A \text{ and } B}{\text{Union of } A \text{ and } B} = \frac{A \cap B}{A \cup B}$$

This is very easily demonstrated using a VENN-diagram:

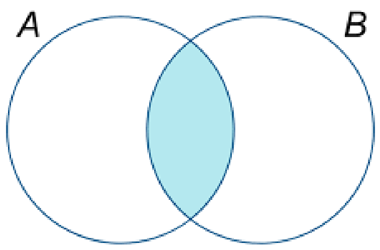


Figure 1: Visualization of Union and Intersection. The Union is the total area covered by both A and B, the Intersection is marked with blue.

For bounding boxes this means we can find how ''good" a bounding box is using the *IoU*, where a value of 1 is a perfect match, 0 is no match.

(b)

When using precision and recall it is normal to use the following abbreviations:

$$\begin{aligned} \text{True Positive} &\rightarrow \text{TP} \\ \text{False Positive} &\rightarrow \text{FP} \\ \text{True Negative} &\rightarrow \text{TN} \\ \text{False Negative} &\rightarrow \text{FN} \end{aligned}$$

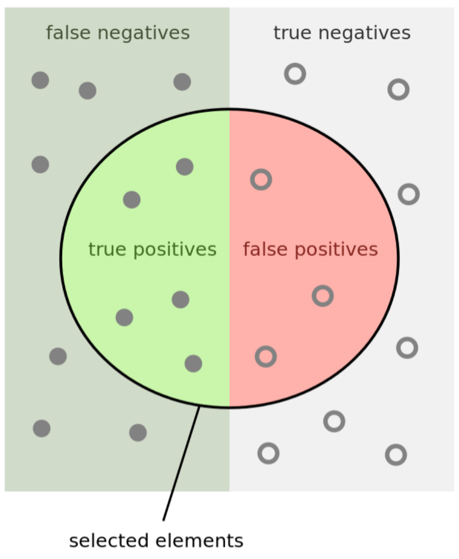


Figure 2: Visualization of the abbreviations above. The circle represents the cases the model assumes to be positive (selected by the model). The left side is actual positive cases, and the right side are actual negative cases.

A true positive (TP) is where the actual case is positive and the model correctly selected it as positive.

A false positive (FP) is where the actual case is negative but the model wrongly selected it as a positive.

Using the abbreviations we get precision and recall is defined as:

$$\begin{aligned} \text{Precision} &= \frac{\text{Number of correct positive predictions}}{\text{Number of positive predictions}} &= \frac{TP}{TP+FP} \\ \text{Recall} &= \frac{\text{Number of correct positive predictions}}{\text{Actual number of positives}} &= \frac{TP}{TP+FN} \end{aligned}$$

(c)

Using the values given in the assignment we interpolate to get the precision values for recall levels [0.0, 0.1, . . . , 1.0]. When interpolating we use the closest value we have to the right of the relevant recall level (ie. to find the value for recall level 0.3, we find the closest known precision for a recall level greater than 0.3, for class 1 this would be recall level 0.4 with $precision = 1.0$). The interpolated values are shown in Table 1.

Table 1: Interpolated Precision and Recall Curves

(a) Class 1

Recall	Precision
0.0	1.0
0.1	1.0
0.2	1.0
0.3	1.0
0.4	1.0
0.5	0.5
0.6	0.5
0.7	0.5
0.8	0.2
0.9	0.2
1.0	0.2

(b) Class 2

Recall	Precision
0.0	1.0
0.1	1.0
0.2	1.0
0.3	1.0
0.4	0.8
0.5	0.6
0.6	0.5
0.7	0.5
0.8	0.2
0.9	0.2
1.0	0.2

Using the Precision and recall curves from Table 1, we first calculate the average precision (AP) for both classes:

$$\text{Class 1: } AP_1 = \frac{\sum_{0.0}^{1.0} \text{Precision}_1}{11} = \frac{1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 0.5 + 0.5 + 0.5 + 0.2 + 0.2 + 0.2}{11} = 0.6454$$

$$\text{Class 2: } AP_2 = \frac{\sum_{0.0}^{1.0} \text{Precision}_2}{11} = \frac{1.0 + 1.0 + 1.0 + 1.0 + 0.8 + 0.6 + 0.5 + 0.5 + 0.2 + 0.2 + 0.2}{11} = 0.6363$$

We can then calculate the mean average precision (mAP, essentially the average of the average precisions) of the two models with:

$$mAP = \frac{AP_1 + AP_2}{2} = \frac{0.6454 + 0.6363}{2} = \underline{\underline{0.64085}}$$

Task 2

All subtasks are implemented in **task2/task2.py**.

(f)

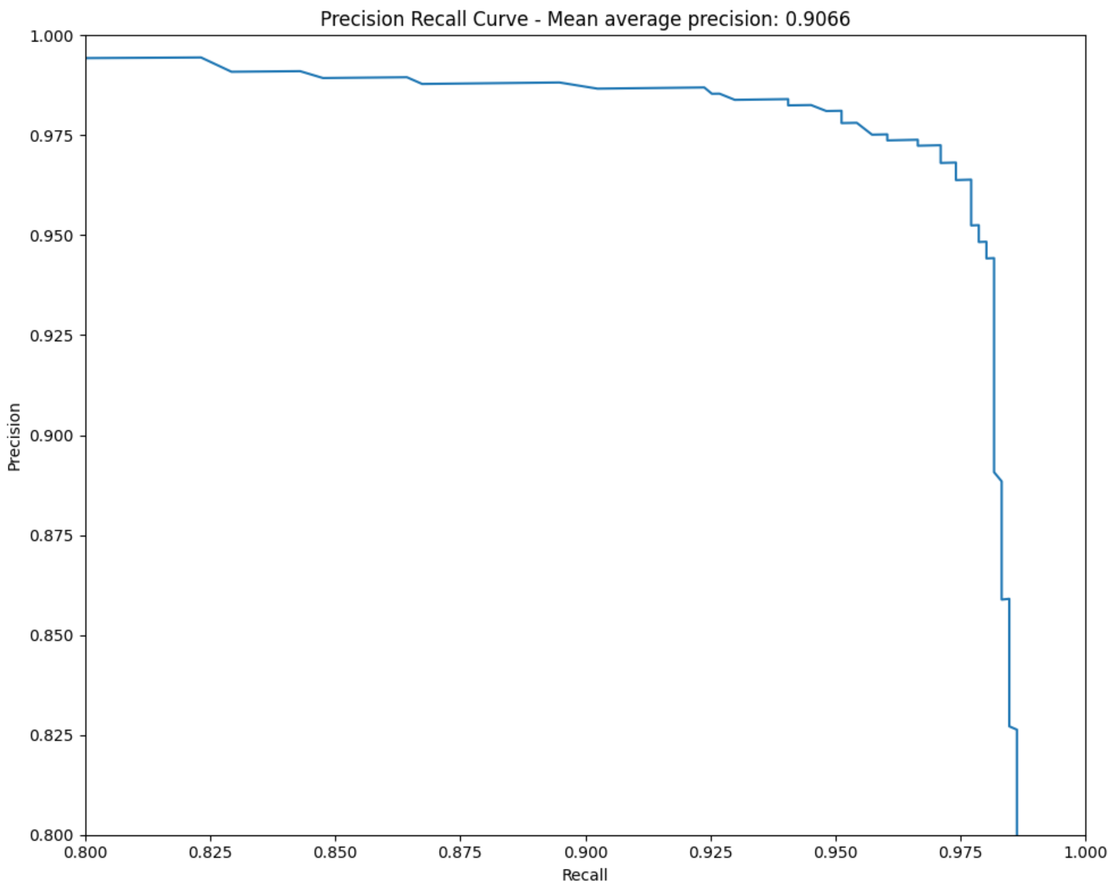


Figure 3: Precision Recall Curve using IoU Threshold of 0.5

Task 3

(a)

When performing inference with SSD, we filter out a set of overlapping boxes using **Non-Maximum Suppression**.

(b)

The statement:

Predictions from the deeper layers in SSD are responsible to detect small objects.

is **False**.

When using SSD, higher-resolution feature maps are responsible for detecting smaller objects.[1]

(c)

They use different bounding box aspect ratios at the same spatial location because different object sizes and shapes are easier detected by some aspect ratios than others.

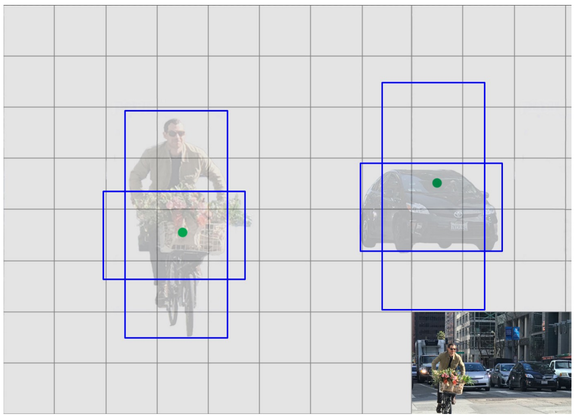


Figure 4: Demonstration how different objects match certain aspect ratios better than others.[2]

[1]SSD Blog Post by Jonathan Hui
[2]SSD Blog Post by Jonathan Hui

(d)

The main difference between *SSD* and *YOLO* is that the *YOLO* architecture consists of fully connected layers while *SSD* only consists of CNN layers. This means that *SSD* is able to classify the same objects at different sizes, while *YOLO* only can classify objects of a certain size.

SSD predicts bounding boxes from several feature maps from the backbone network. YOLO V1/v2 does not do this.

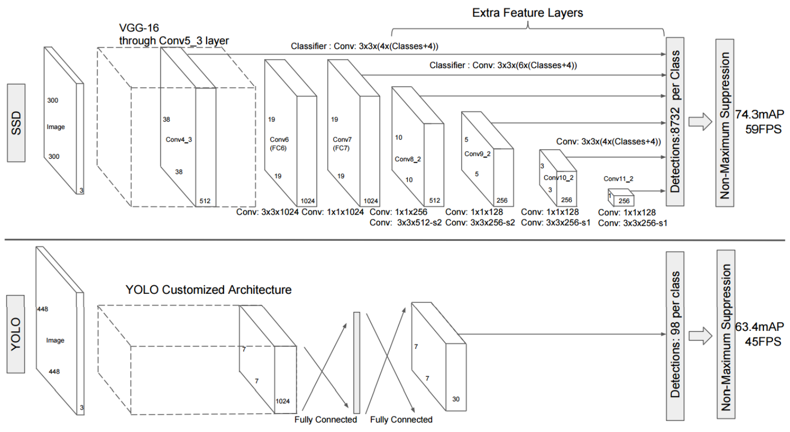


Figure 5: A comparison between SSD and YOLO network architecture[3]

(e)

Because the feature maps is of size 38×38, we have $38 \cdot 38$ anchor points, and we have 6 anchors, this means that for this feature map we have:

$$38 \cdot 38 \cdot 6 = \mathbf{8664} \text{ anchor boxes}$$

(f)

The total number of anchor boxes for this network is:

$$\begin{array}{lllllll} 38 \cdot 38 \cdot 6 & +19 \cdot 19 \cdot 6 & +10 \cdot 10 \cdot 6 & +5 \cdot 5 \cdot 6 & +3 \cdot 3 \cdot 6 & +1 \cdot 1 \cdot 6 & = \\ 8664 & +2166 & +600 & +150 & +54 & +6 & = \mathbf{11\ 640} \end{array}$$

[3]SSD: Single Shot MultiBox Detector.

Task 4

I included a makefile in the assignment_code.zip (SSD/makefile) that contains the commands i used for each of the following subtasks.

(a)

The model is implemented in **SSD/ssd/modeling/backbones/basic.py**.

(b)

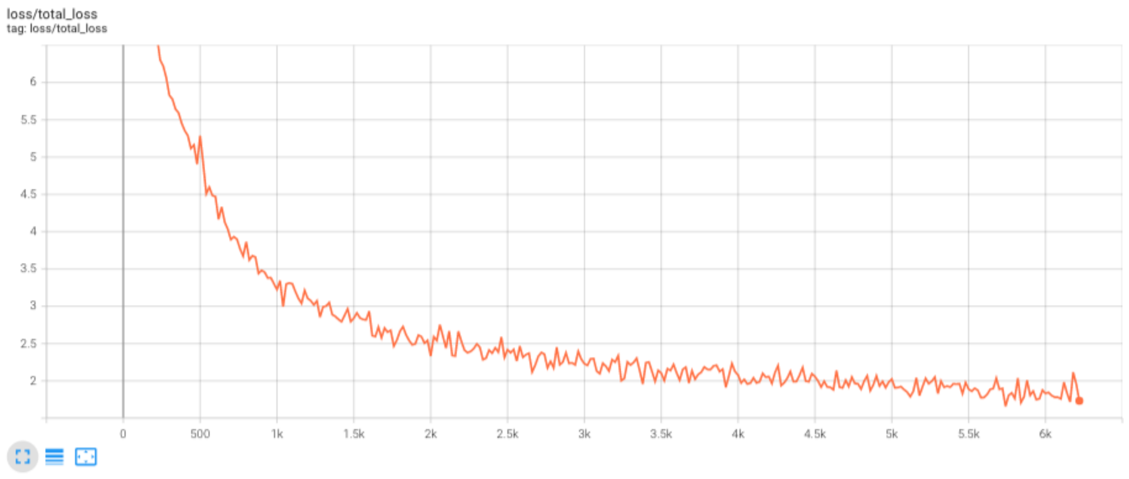


Figure 6: Total Loss for model from Task 4a over 20 epochs (6240 iterations).

The final result of the model was:

```
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.74065
```

In other words the model achieves **Mean Average Precision ≈ 74%**

(c)

To improve the model I:

- changed the filter sizes and padding for a few layers
- increased number of filters for some layers
- changed the optimizer and the learning rate
- added batch normalization and dropout
- changed the normalization values used on the data

The model is implemented in **SSD/ssd/modeling/backbones/basic_improved.py**, and the hyperparamters and normalization were changed in the file **SSD/configs/ssd300_improved.py**.

The complete model is shown in Table 2 and the hyperparameters used are listed in Table 3.

Table 2: The Improved Model. Using output_channels = [256, 512, 256, 128, 64, 64]

Is Output	Layer Type	Number of Filters	Kernel Size	Stride	Padding
	Conv2d	32	5	1	2
	BatchNorm2d	-	-	-	-
	Dropout(0.1)	-	-	-	-
	ReLU	-	-	-	-
	MaxPool2d	-	2	2	-
	Conv2d	64	5	1	2
	BatchNorm2d	-	-	-	-
	Dropout(0.1)	-	-	-	-
	ReLU	-	-	-	-
	MaxPool2d	-	2	2	-
	Conv2d	128	5	1	3
	BatchNorm2d	-	-	-	-
	Dropout(0.1)	-	-	-	-
	ReLU	-	-	-	-
	Conv2d	output_channels[0]	5	2	2
	BatchNorm2d	-	-	-	-
Yes - Resolution 38×38	ReLU	-	-	-	-
	ReLU	-	-	-	-
	Conv2d	256	5	1	2
	ReLU	-	-	-	-
	Conv2d	output_channels[1]	5	2	2
Yes - Resolution 19×19	ReLU	-	-	-	-
	ReLU	-	-	-	-
	Conv2d	512	5	1	2
	ReLU	-	-	-	-
	Conv2d	output_channels[2]	5	2	2
Yes - Resolution 10×10	ReLU	-	-	-	-
	ReLU	-	-	-	-
	Conv2d	256	3	1	1
	ReLU	-	-	-	-
	Conv2d	output_channels[3]	3	2	1
Yes - Resolution 5×5	ReLU	-	-	-	-
	ReLU	-	-	-	-
	Conv2d	128	3	1	1
	ReLU	-	-	-	-
	Conv2d	output_channels[4]	3	2	1
Yes - Resolution 3×3	ReLU	-	-	-	-
	ReLU	-	-	-	-
	Conv2d	128	3	1	1
	ReLU	-	-	-	-
	Conv2d	output_channels[5]	3	1	0
Yes - Resolution 1×1	ReLU	-	-	-	-

Table 3: Hyperparameters for the improved model.

Hyperparameter	value
Optimizer	*Adam*
Batch Size	32
Learning Rate	0.0005

The **mean** and **std** values used for normalization fo the data were changed to the ones used in the previous assignment:

- **mean** $= (0.485, 0.456, 0.406)$
- **std** $= (0.229, 0.224, 0.225)$

The loss of the model is seen in Figure 7, and the Mean Average Precision is shown in Figure 8.

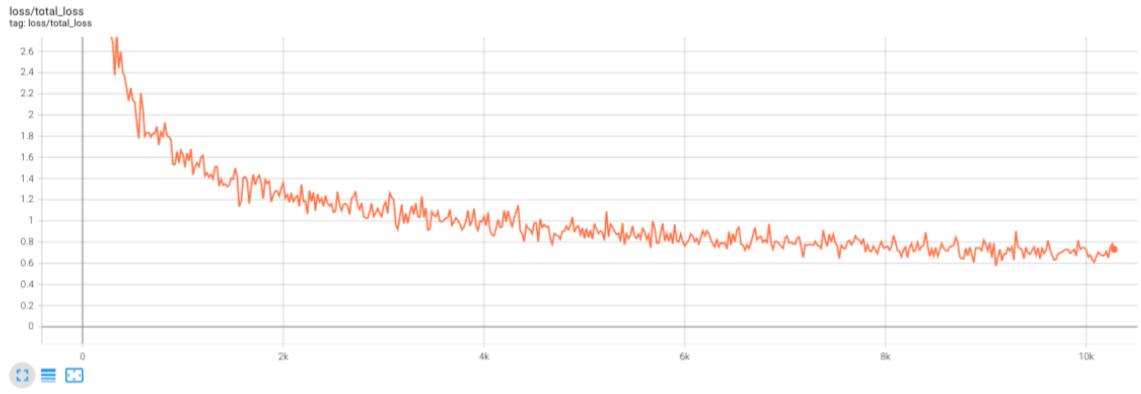


Figure 7: Loss from my improved model

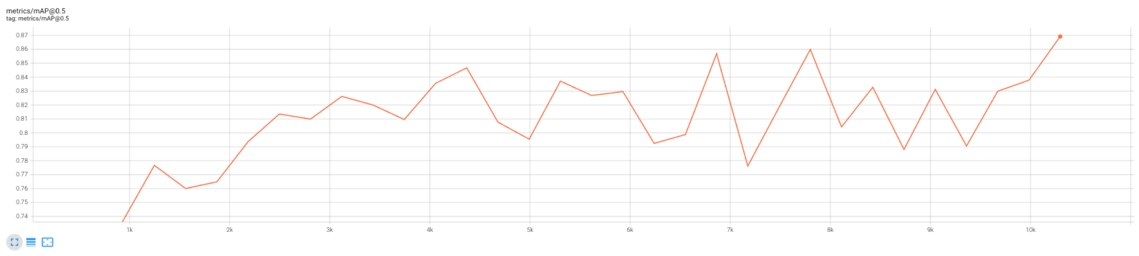


Figure 8: Mean Average Precision for my improved model

As we can see it reached a $mAP > 85\%$ three times within the 10K iterations, these are listed in Table 4.

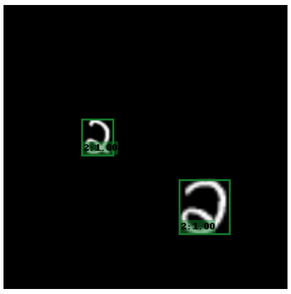
Table 4: Instances where the mAP was above 85% for the model.

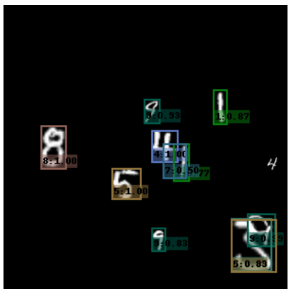
Steps	mAP
6 864	85.68%
7 800	85.99%
10 296	86.92%

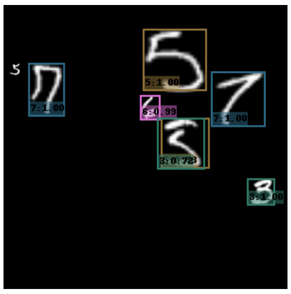
The final Mean Average Precision of the model with IoU Threshold = 0.5 after the full 33 epochs (10 296 iterations) was **86.92%**:

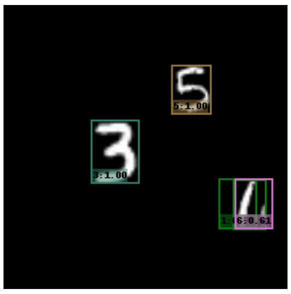
```
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.86920
```

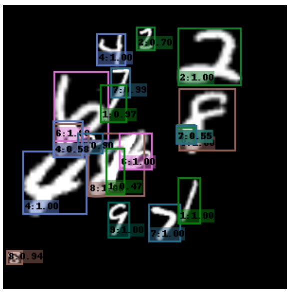
(e)

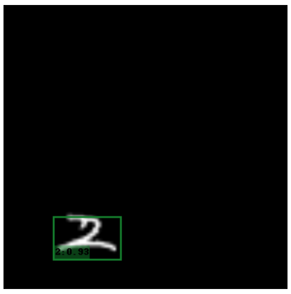

(a) 0.png

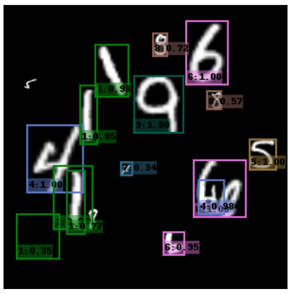

(b) 1.png

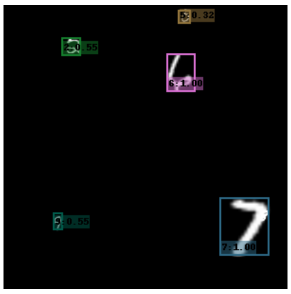

(c) 2.png

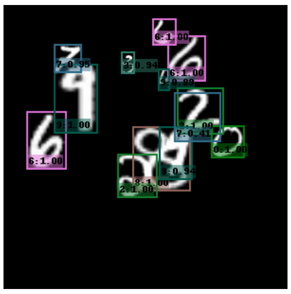

(d) 3.png

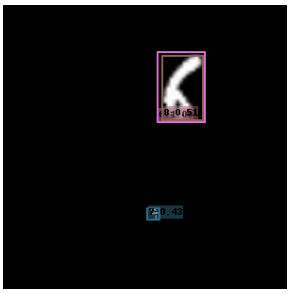

(e) 4.png

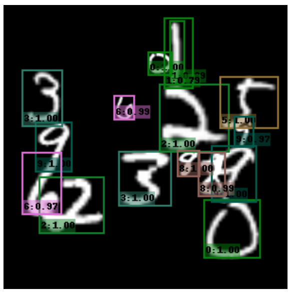

(f) 5.png

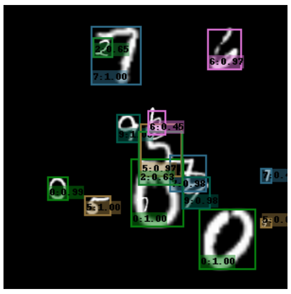

(g) 6.png

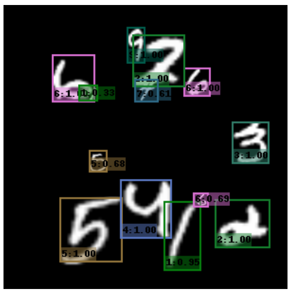

(h) 7.png

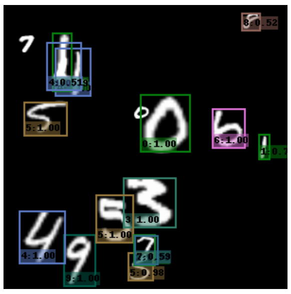

(i) 8.png


(j) 9.png


(k) 10.png


(l) 11.png


(m) 12.png


(n) 13.png

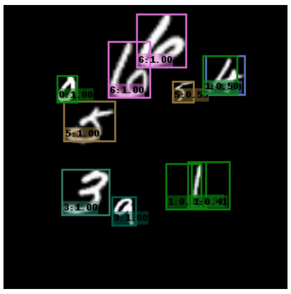
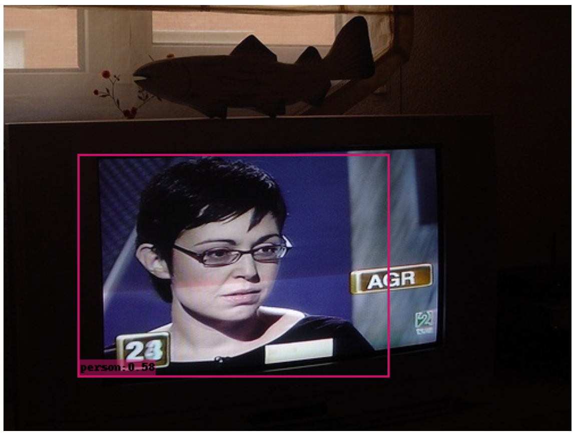

(o) 14.png

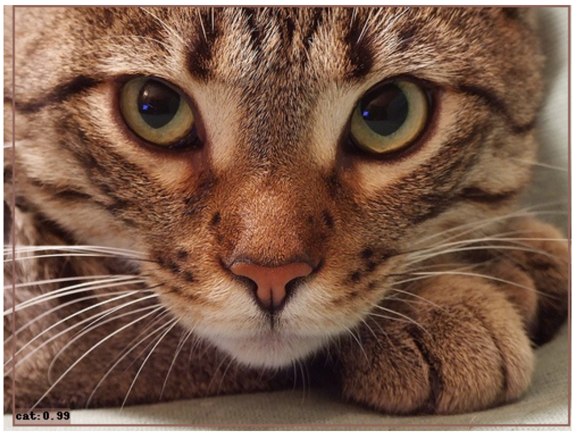
Figure 9: The output of using the model trained in Task 4c on the images in the **SSD/demo/mnist** folder.

As we can see in Figure 9 there are a few instances where my model isn't able to detect a number (such as in (b), (c), (g) and (n)). There are also instances where my model finds numbers that aren't there, or where it finds the same number several times (as can for instance be seen in (b), (c), (d), (e), (g), (i), (j), (k) and (o)). It seems my model is overly eager to classify things which leads to a lot more false positives than false negatives.

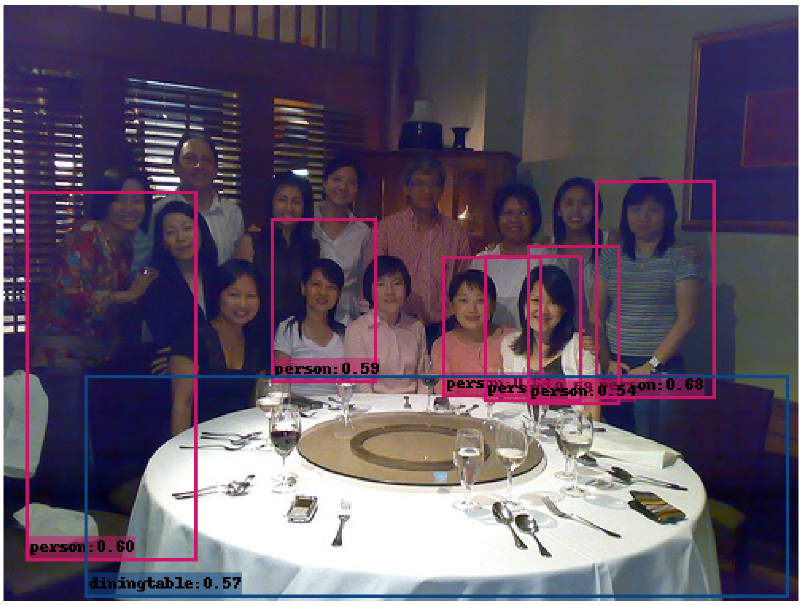
(f)



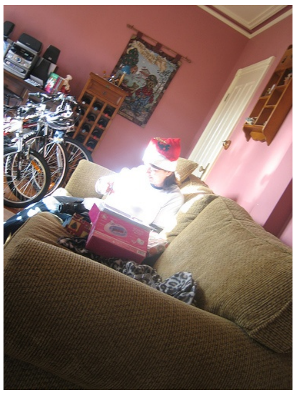
(a) 000342.png



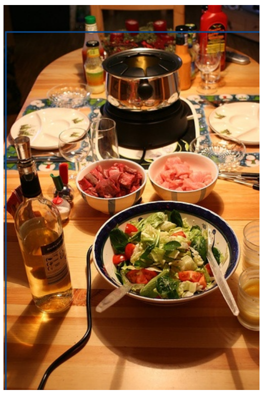
(b) 000542.png



(c) 003123.png



(d) 004101.png



(e) 008591.png

Figure 10: The output after training the pre-trained VGG16 model on the voc dataset for 5000 iterations, then using the model on the images in the **SSD/demo/voc** folder.

The loss of the model during training is seen in Figure 11, and the Mean Average Precision is shown in Figure 12.

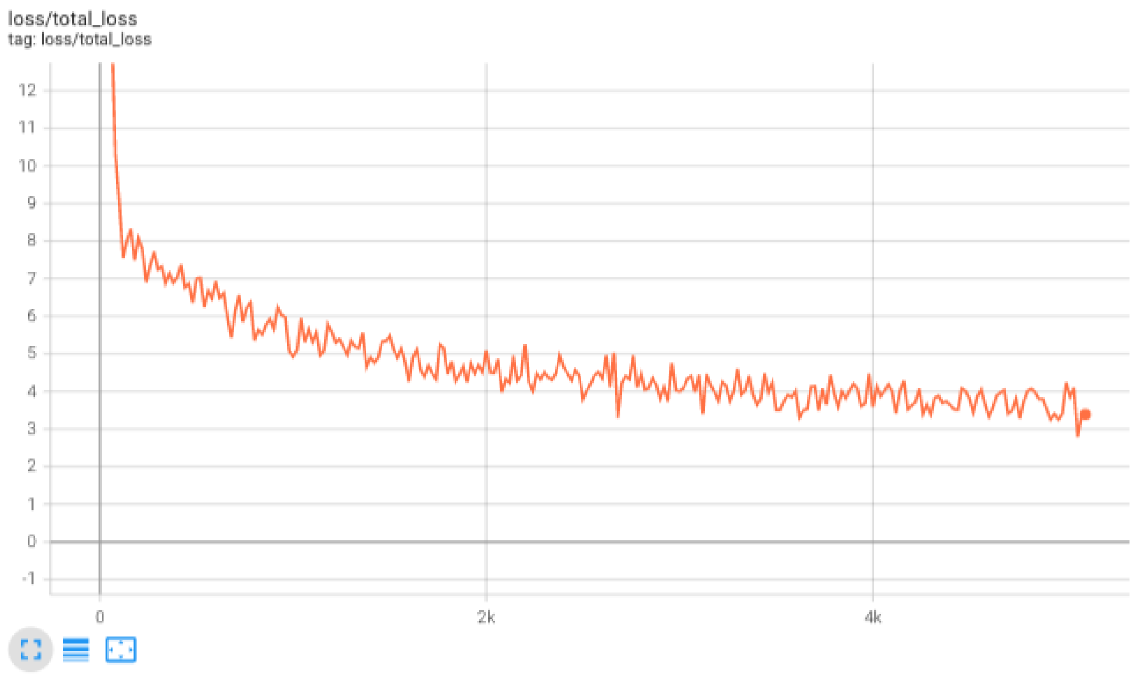


Figure 11: Loss during training of VGG16

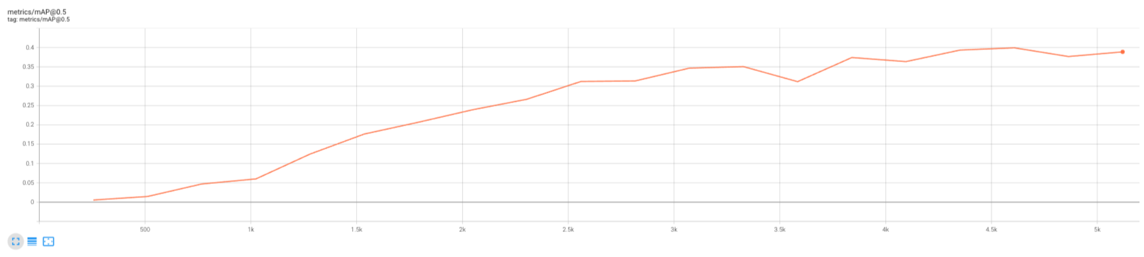


Figure 12: Mean Average Precision for VGG16 during training