NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4265 - Computer Vision and Deep Learning

Assignment 4

Kolbjørn Kelly
Sander Endresen

24th March 2021

Contents

1 Task 1: Object Detection Metrics

1.1 a) Intersection over Union

The Intersection over Union (IoU) is how much the area of the prediction overlaps with the area of the ground truth. It means how big part of the prediction boundary box that matches the boundary box of the real object. An illustrative example can be seen in Figure 1 and 2. Its expression is given by the following, where A_p denotes the predicted area and A_{gt} the area of the ground truth:

$$IoU = \frac{A_p \cap A_{gt}}{A_p \cup A_{gt}} = \frac{\text{overlapped area}}{\text{area of union of ground truth and prediction}}$$

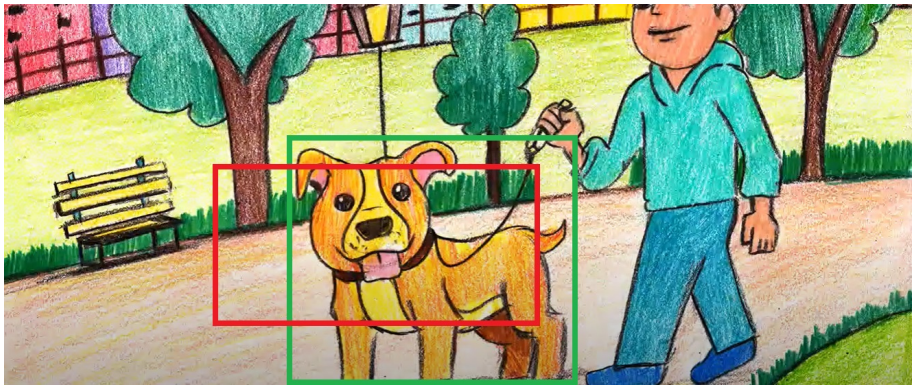

Figure 1: 1a) Green boundary box: ground truth, red boundary box: prediction

Figure 2: 1a) Dark green area (D): overlap between ground truth and prediction, light green area (L): union of ground truth and prediction

In the example above the Intersection over Union is given as $\frac{D}{D+L}$.

1.2 b) Precision, recall and true and false positives

Precision is the amount of correct predictions of all predictions. One can express it as

$$\text{precision} = \frac{TP}{TP + FP} = \frac{\text{True Positives}}{\text{All detections}},$$

where TP = True Positives and FP = False Positives.

Recall is the proportion of objects detected. It shows how much of the objects that can be detected, actually are detected. Its expression is given as

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{True Positives}}{\text{All ground truths}},$$

where TP = True Positives and FN = False Negatives.

To distinguish between correct and wrong detections for out predictions, we use a threshold for the IoU. If we have $IoU \geq threshold$ we say that it is a correct detection, and thereby a true positive. When $IoU < threshold$, it is a wrong detection and we say that it is a false negative. In other words, a true positive occurs when a correct prediction has been made, and a false negative occurs when there is an absence of a prediction.

1.3 c) Mean average precision

We have the following precision and recall curves

Class 1:
precision: [1.0, 1.0, 1.0, 0.5, 0.20]
recall: [0.05, 0.1, 0.4, 0.7, 1.0]

Class 2:
precision: [1.0, 0.80, 0.60, 0.5, 0.20]
recall: [0.3, 0.4, 0.5, 0.7, 1.0]

and the following calculations were implemented as a Python script:

$$\text{AP} = \frac{1}{11} \sum_{r} \max_{\hat{r};\hat{r}>r} p(\hat{r})$$

$$\text{mAP} = \frac{1}{N} \sum_{i}^{N} \text{AP}_i$$

The function **calculate_mean_average_precision** from task 2e) was used, and the redeclaration of the recall levels were changed as shown below. The reason for this change is that the two declarations of *recall_levels* gave different results for the APs when running the Python script and **calculate_mean_average_precision**, which should have given the same results. This may come from errors according to floating points or something else with the *linspace* function, compared to array, using numpy. By redeclaring *recall_levels* as shown below they both gave the same results.

```
#old declaration
recall_levels = np.linspace(0, 1.0, 11)


#new declaration
recall_levels = np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
```

The mean average precision (mAP) for the two classes were found the following way:

1. Running the function **calculate_mean_average_precision** with the new declaration of *recall_levels*, with the precision and recall for class 1 and 2 as inputs. From here we get the average precision for each class as the respectively outputs *average_precision_1* and *average_precision_2*.

2. Taking the mean of the two average precisions by running
   (*average_precision_1* + *average_precision_2*)/2.

With the given values, we got *average_precision_1* = 0.6455 and *average_precision_2* = 0.6364, which gives a mAP of 0.6409.

2 Task 2: Implementation of Mean Average Precision

2.1 f) Final precision-recall

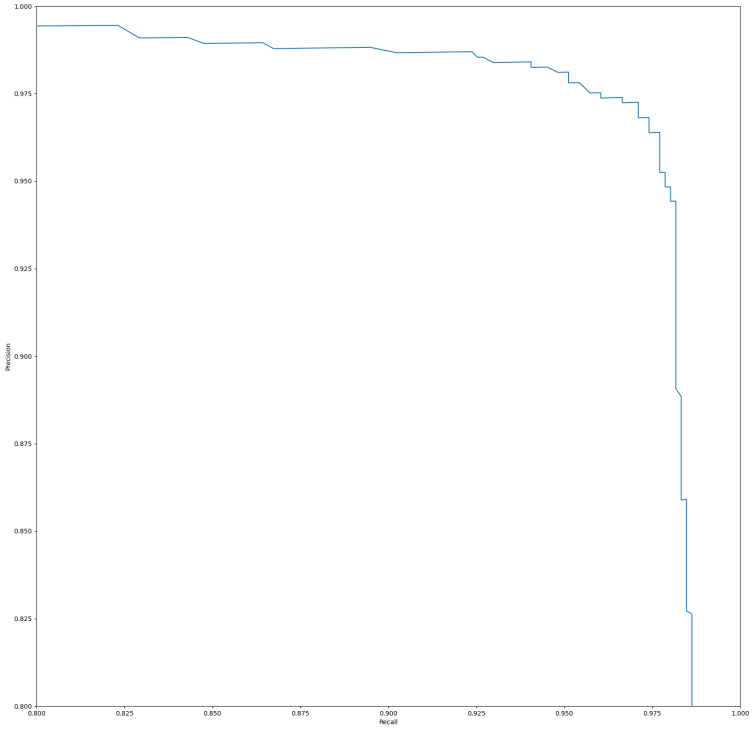

Figure 3: Final PR-curve

3 Task 3: Theory

3.1 a) Filtering operation

The filtering operation to filter out a set of overlapping boxes is called non-maximum suppression.

3.2 b) Detection of smaller objects

The given statement is **FALSE**, because the high-resolution feature maps, where Conv4_3 layer is the first layer for object detection, is responsible for detecting small objects. Information will

be lost in the deeper layers in the architecture of SSD, because the CNNs reduce the spatial dimensions. Thereby it is harder to make out the smaller objects. To detect objects at larger scales the layers with lower resolution are utilized, and the high resolution layers are utilized for small objects.

3.3 c) Bounding box aspect ratios at the same spatial location

Different bounding box aspect ratios is used at the same spatial location to be able to detect multiple objects at the same location. E.g. a person and a car may be on the same location of an image or overlap, but because a person is tall and thin and the car is short and wide, the aspect ratios of the bounding boxes are different. To be able to detect all objects we therefore need more than one bounding box aspect ratio.

3.4 d) SSD vs. YOLOv1/v2

The main difference between the SSD and YOLO architectures is that two fully connected layers are used in YOLO but convolutional layers of varying size is used in SSD. In SSD the anchor boxes are highly overlapped, which they are not in YOLO. YOLOv1 differ from SSD, where it uses bounding boxes for framing the objects. YOLOv2 is more similar to SSD in terms of prediction of class and bounding boxes, where it also uses anchor boxes.

3.5 e) Anchor boxes for feature map

The last feature map is 38×38 and each anchor location has 6 anchors. Then the total number of anchor boxes for this feature map is

$$38 \cdot 38 \cdot 6 = 8\ 664.$$

3.6 f) Anchor boxes for the entire network

We have the total number of anchor boxes in the network given as $\mathrm{N_{ab}}$ below. By inserting for the given resolutions, we get

$$\begin{aligned}
\mathrm{N_{ab}} &= 6 \cdot \Big(\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{H_i} \times \mathrm{W_i} \Big) \\
&= 6 \cdot \Big[(38 \cdot 38) + (19 \cdot 19) + (10 \cdot 10) + (5 \cdot 5) + (3 \cdot 3) + (1 \cdot 1) \Big] \\
&= 11\ 640
\end{aligned}$$

4 Task 4: Implementation of Single Shot Detector

4.1 b) Our Very First SSD

The final mAP after 6000 steps was 0.7569. A plot of the total loss is shown in figure 4.

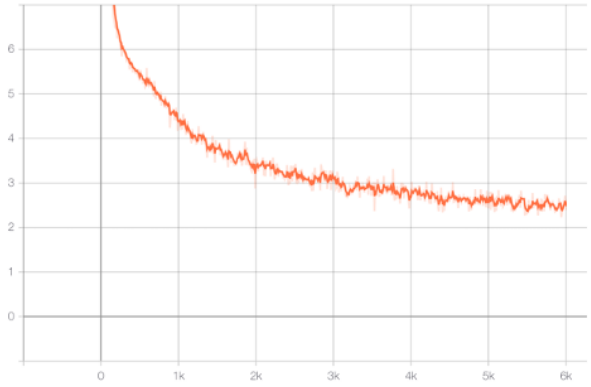

Figure 4: 4b) Total loss

4.2 c) Improvements by Old Techniques

Firstly, spacial batch normalization was added before every ReLU succeeding a convolutional. Secondly, a very conservative dropout was added after the pooling layers. This yielded the following structure

Is Output	Layer Type	Number of Filters	Stride
	Conv2D	32	1
	MaxPool2D	-	2
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2D	64	1
	MaxPool2D	-	2
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2d	64	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 38x38	Conv2D	output_channels[0]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Conv2D	128	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 19x19	Conv2D	output_channels[1]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Conv2D	256	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 9x9	Conv2D	output_channels[2]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Conv2D	128	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 5x5	Conv2D	output_channels[3]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Conv2D	128	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 3x3	Conv2D	output_channels[4]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Conv2D	128	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 1x1	Conv2D	output_channels[5]	1

Also, the Adam optimizer was employed, leaving the learning rate unchanged. As the network seemed to "peak" after about 5k iterations, a scheduler was added to decrease the learning rate by half every 5k iterations. This yielded an mAP of 0.8681 after 10k iterations. The implementation is found in improved_basic.py.

4.3 d) Further Improvements

Firstly, we increased the number of channels in some of the layers, as well as applying dropout to different places (before the last convolutional layer of each feature layer). This yielded the following network

Is Output	Layer Type	Number of Filters	Stride
	Conv2D	32	1
	MaxPool2D	-	2
	ReLU	-	-
	Conv2D	64	1
	MaxPool2D	-	2
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2d	128	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 38x38	Conv2D	output_channels[0]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2D	256	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 19x19	Conv2D	output_channels[1]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2D	512	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 9x9	Conv2D	output_channels[2]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2D	256	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 5x5	Conv2D	output_channels[3]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2D	128	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 3x3	Conv2D	output_channels[4]	2
	BatchNorm2D	-	-
	ReLU	-	-
	Dropout(0.1)	-	-
	Conv2D	128	1
	BatchNorm2D	-	-
	ReLU	-	-
Yes - 1x1	Conv2D	output_channels[5]	1

Furthermore, the Jaccard threshold was increased from 0.5 to 0.6. Additionally the minimum size of the prior boxes from the first feature layer was decreased from 30×30 to 15×15.

This yielded a final mAP of 0.9050. Plots of the mAP and total loss is shown in figure 5 and 6. The implementation is found in final_model.py.

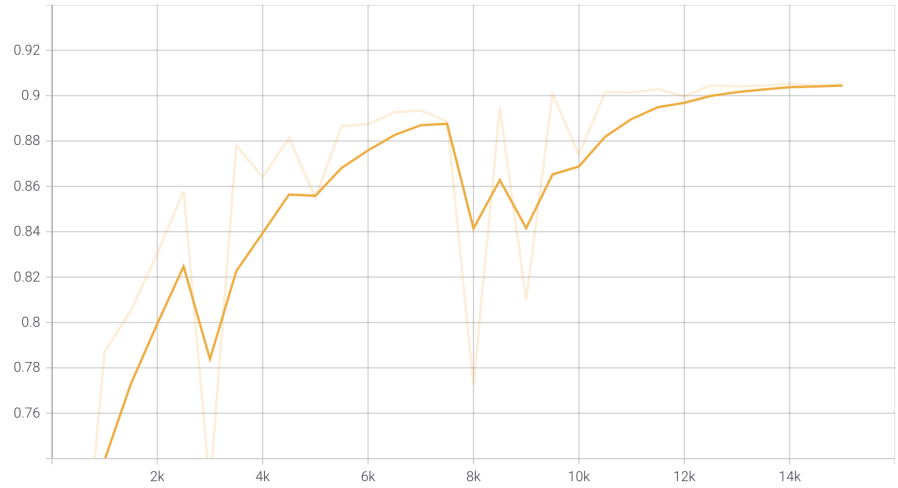

Figure 5: 4d) mAP

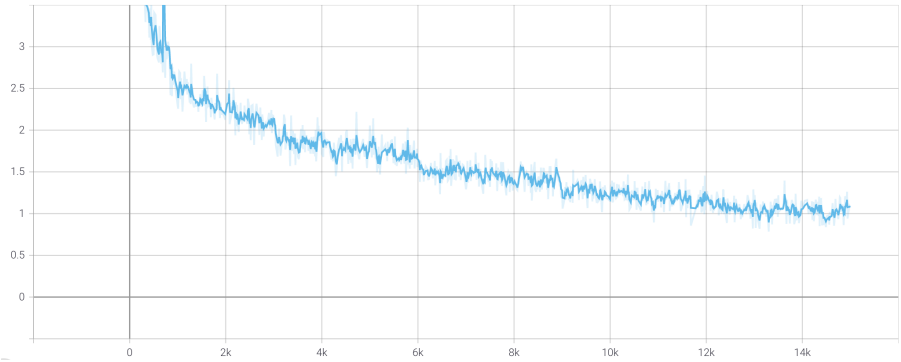

Figure 6: 4d) Total Loss

4.4 e) Model testing

The results are shown in figure 7. As expected, the model struggles to detect small digits and ones. The reason for the first might be that the resolution of 38×38 at the first feature level is too low. The reason for the second might be that ones fits better in narrower rectangles than the other digits. Scaling prior boxes could probably improve this.

Figure 7: 4e) MNIST Detection results

4.5 f) Pascal VOC

The final mAP was 0.1571. The final results are shown in figure 8, and a plot of the final loss is shown in figure 9. As can be seen, the cat is the only prediction made.

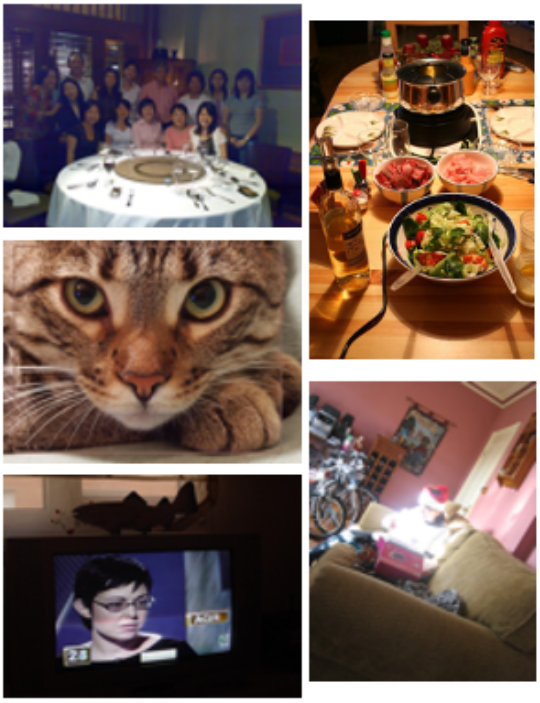

Figure 8: 4f) VOC results

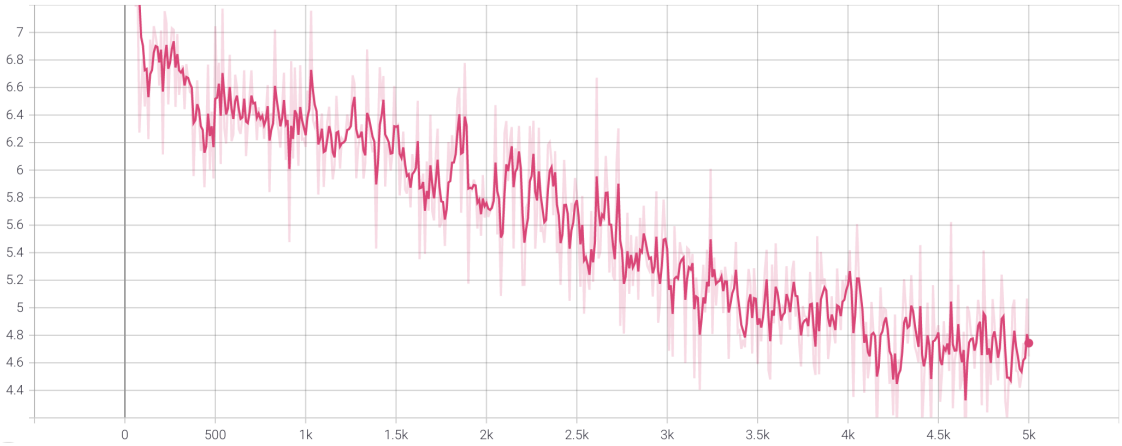

Figure 9: 4f) Total Loss