NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

TDT4265 - COMPUTER VISION AND DEEP LEARNING

Assignment 4

Authors
Eivind STRAY
Fridtjof HØYER

March 24, 2021

NTNU
Norwegian University of
Science and Technology

1 Task 1

1.1 task 1a

When working with object detection bounding boxes are used to identify objects. Intersection over Union is a way of evaluating the accuracy of the detector once it is trained.

For calculating the intersection over union one has two different bounding boxes, one that shows the true boxes to indicate where the object is and one showing the predicted bounding box. the image below shows an example of a predicted bounding box and its ground-truth bounding box.

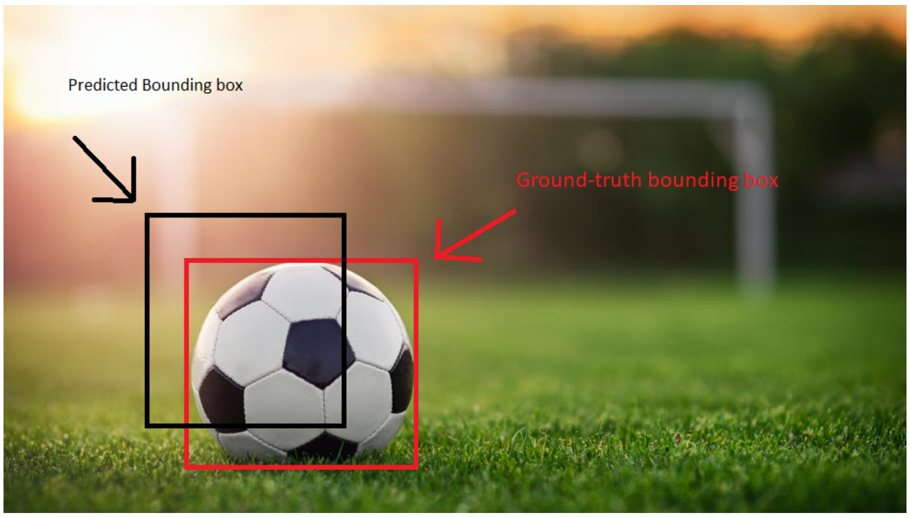


Figure 1: Predicted bounding box and its ground-truth bounding box.

To calculate the intersection over union we need to find the Area of overlap and the area of union. The images below shows how to find these values.

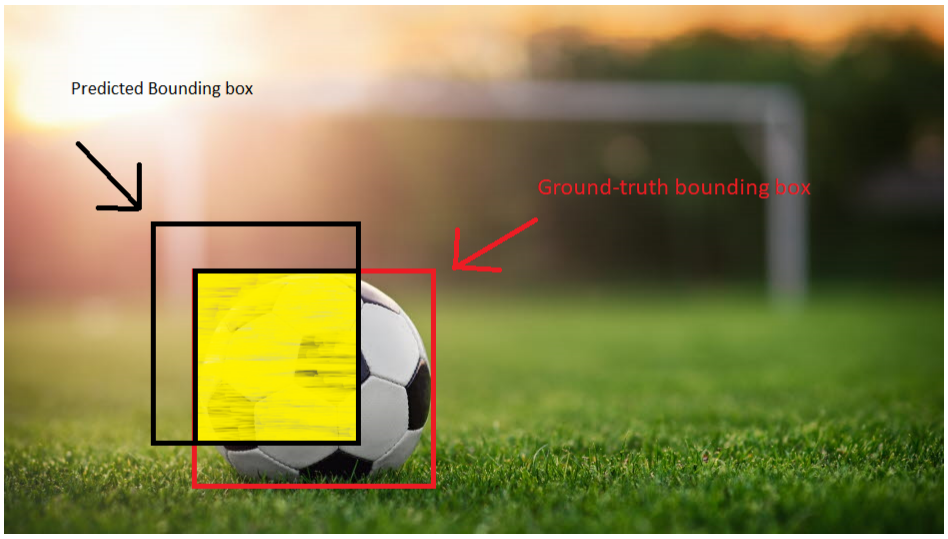


Figure 2: Area of Overlap

Intersection over Union is calculated by dividing the Area of overlap with the Area of Union.

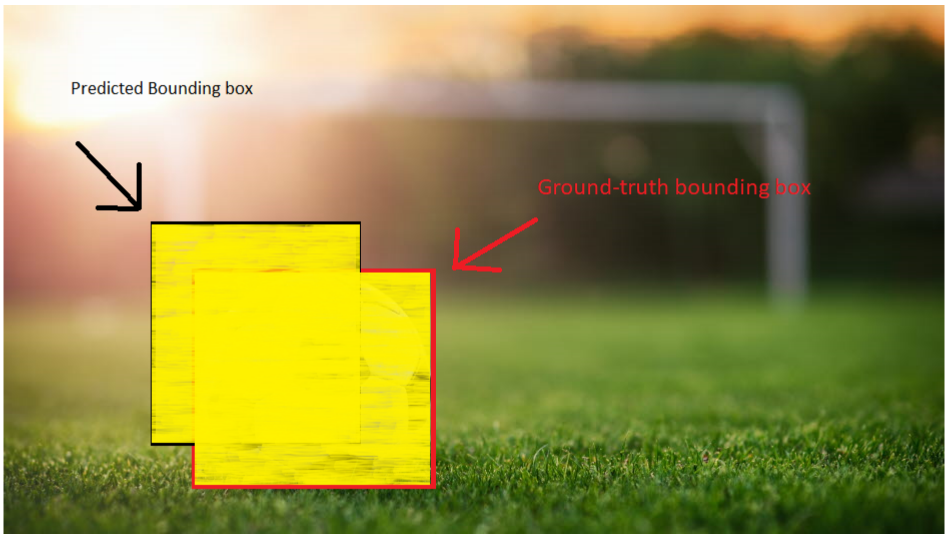


Figure 3: Area of union.

$$\text{Intersection over Union} = \frac{\text{Area of overlap}}{\text{Intersection over Union}}$$

1.2 task 1b

True positive and false positive is easy to explain by a binary example.

Let's say you are tested for the coronavirus. A True positive is a case where you have corona and the test indicates that you have corona.

A false positive is a case where you do not have coronavirus but the test indicates that you have corona.

Example: True positive: If a car is classified as a car.
False positive: If a butterfly is classified as a car.

Precision $= \frac{TruePositive}{TruePositive+FalsePositive}$

In the coronavirus example: What is the proportion of positive tests that are correct?

Recall $= \frac{TruePositive}{TruePositive+FalseNegative}$

In the coronavirus example: What is the proportion of actual coronavirus cases that are detected?

1.3 Task 1c

First we need to calculate the AP for the given Precision and Recalls.

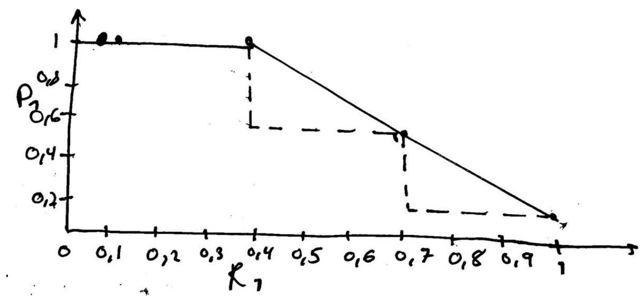


Figure 4: First precision recall plot

$$AP_1 = \frac{5 * 1 + 3 * 0,5 + 3 * 0,2}{11} = 0,645 \tag{1}$$

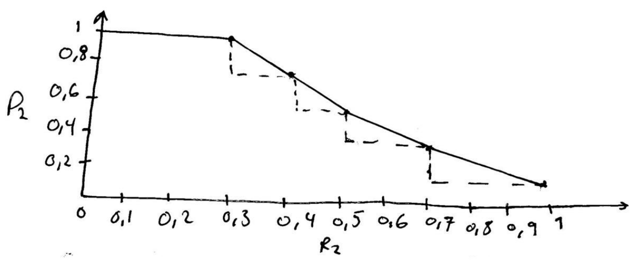


Figure 5: Second precision recall plot

$$AP_2 = \frac{4 * 1 + 1 * 0,8 + 1 * 0,6 + 2 * 0,5 + 3 * 0,2}{11} = 0,636 \tag{2}$$

Then we get that the mAP is equal:

$$mAP = \frac{AP_1 + AP_2}{2} = 0,641 \tag{3}$$

2 Task 2

2.0.1 task 2e

The mAP ended up at the expected 0.9066, and the precision-recall curve can be found below in Figure 6:

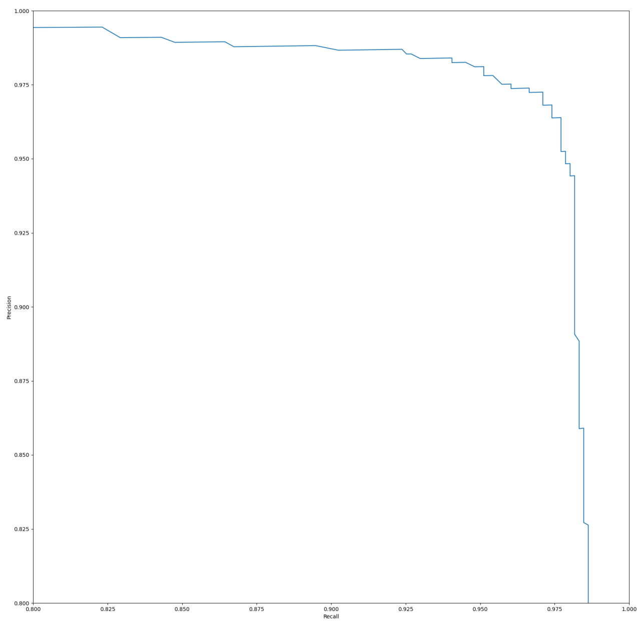


Figure 6: Precision-recall curve.

3 Task 3

3.1 task 3a

The process of filtering out redundant overlapping boxes for an object is called non-maximum suppression.

3.2 task 3b

As we move deeper into the network, the resolution gets lower and lower. Lower resolution feature maps is used to detect larger objects, so the shallower layers (left) detect small objects. From this, the claim that the deeper layers (closer to the output) detect small object is False.

3.3 Task 3c

One of the reasons SSD use different box aspect ratios at the same location is to find boundary box predictions for multiple objects. Only using one bounding box aspect ratio works find when only trying to find one particular object. If the predictions cover more shapes the model can detect more types of objects. The boundary boxes are often clustered to find the width and height of the boundary boxes. Conceptually, the ground truth boundary boxes can be partitioned into clusters with each cluster represented by a default boundary box. The reason for this is to not make random guesses, but make guesses based on defualt boundary boxes.

3.4 Task 3d

SSD adds convolutional future layers at the end of the truncated base network. This means that SSD uses a multi-scale future map. These future layers decreases the size and allows prediction of detection at multiple size. YOLOv1/v2 on the other only use single scale future map.

3.5 Task 3e

For calculating the number of anchor boxes we simply calculate:

$$\text{Number of anchor boxes} = H \cdot W \cdot k$$

Where k is the number of anchors.

We get:

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

3.6 Task 3f

As the last subtask we calculate the number of anchors in the network by calculating $H \cdot W \cdot k$ for each layer in the newtwork:

We get:

$$38 \cdot 38 \cdot 6 + 19 \cdot 19 \cdot 6 + 10 \cdot 10 \cdot 6 + 5 \cdot 5 \cdot 6 + 3 \cdot 3 \cdot 6 + 1 \cdot 1 \cdot 6 = 11640 \text{ anchor boxes.}$$

4 Task 4

4.1 task 4a and b

Training 6000 iterations resulted in a mAP of 77% and 10 000 at 0.81%. We produced the plots by training 10 000 iterations.

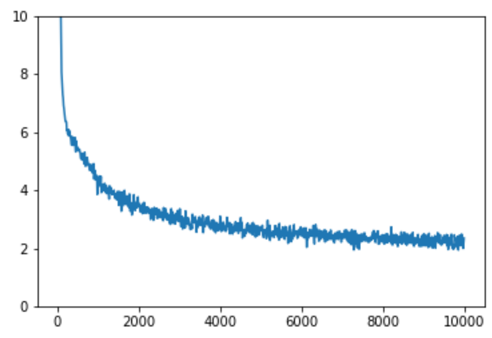


Figure 7: Total Loss - 10 000 iterations "basic network".

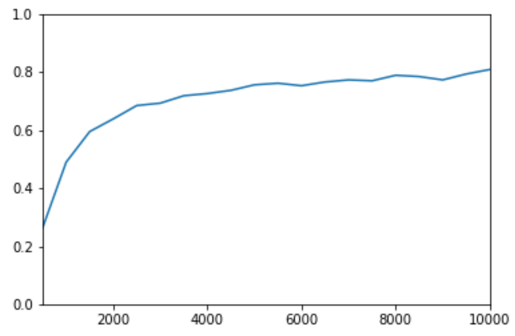


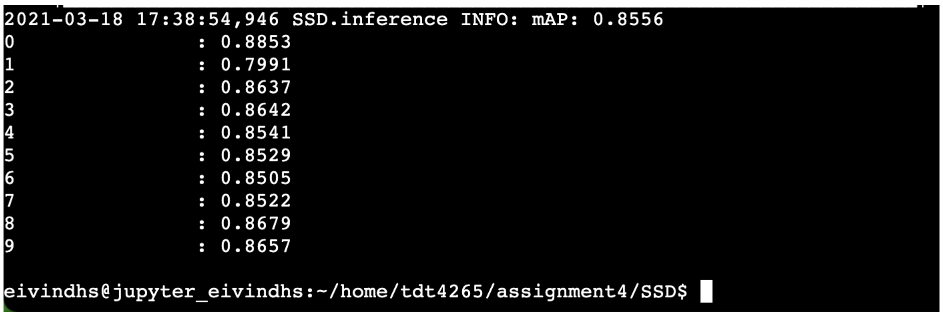
Figure 8: mAP - 10 000 iterations, "basic network".

4.2 task 4c

In this exercise, our goal is to improve the model from 4a and b such that the mAP ends up at more than 85% after 10 000 iterations. We tried a few different things, but ended up with a batch normalization after the last convolution in every layer as well as layer 2-6 consisting of three filters rather than two. Each convolution operation in these layers are followed by batch normalization. Furthermore, we used the Adam optimizer and a learning rate of 10^{-3}. The exact implementation can be found in "improved.py". This gave good results, but quite unstable training.

We then tried using the original basic network, but with batch normalization after every convolution operation as well as using the Adam optimizer. Furthermore, we increased the learning rate again to $2 * 10-3$

As this did not lead to more than 84%, we continued trying. We changed all ReLUs apart from the last in every layer with the activation function "LeakyReLU" which essentially becomes somewhat negative if the input is less than zero, as apposed to ReLU which is exactly zero. We also decreased the learning rate to $7 \cdot 10^{-4}$. This did in fact reach more than 85.6% mAP after 10 000 iterations.



```
2021-03-18 17:38:54,946 SSD.inference INFO: mAP: 0.8556
0               : 0.8853
1               : 0.7991
2               : 0.8637
3               : 0.8642
4               : 0.8541
5               : 0.8529
6               : 0.8505
7               : 0.8522
8               : 0.8679
9               : 0.8657

eivindhs@jupyter_eivindhs:~/home/tdt4265/assignment4/SSD$
```

Figure 9: Proof of more than 85% - 10 000 iterations.

4.3 task 4d

By increasing the number of filters in layer 1-6, as well as increasing the confidence threshold to 0.7 and a learning rate of $1 \cdot 10^{-3}$, we increased the mAP further and it reached a maximum of 88.8%. This did, however, make the training way less stable, and the mAP ended up at about 66%. The exact implementation can be found in "improved_more.py"

4.4 task 4e

The full demo results can be found in Figure 10. It seems the detector has problems detecting the smallest numbers, but none seem to be favored over others. We were expecting the number "1" to be misclassified frequently as other numbers, but this seems not to be case from this demo. The model used is the "improved" - yielding a mAP of 85.6% (described in Section 4.2).

4.5 task 4f

In Figure 12, you can see the results with the requested network trained for 5000 iterations. Here, the score threshold is set to 0.5, as none of these have a score of more than 0.7 yielding the first, unmodified output useless (it's the same as the original images). This would probably get better and better with more training, as the training is probably not even close to converging. The total Loss throughout training over 5000 iterations can be found in Figure 11. The final mAP after 5000 iterations ended up being 0.2640, or 26.4%.

This was done before the assignment correction.

4.6 Images and plots for Task 4 e and f

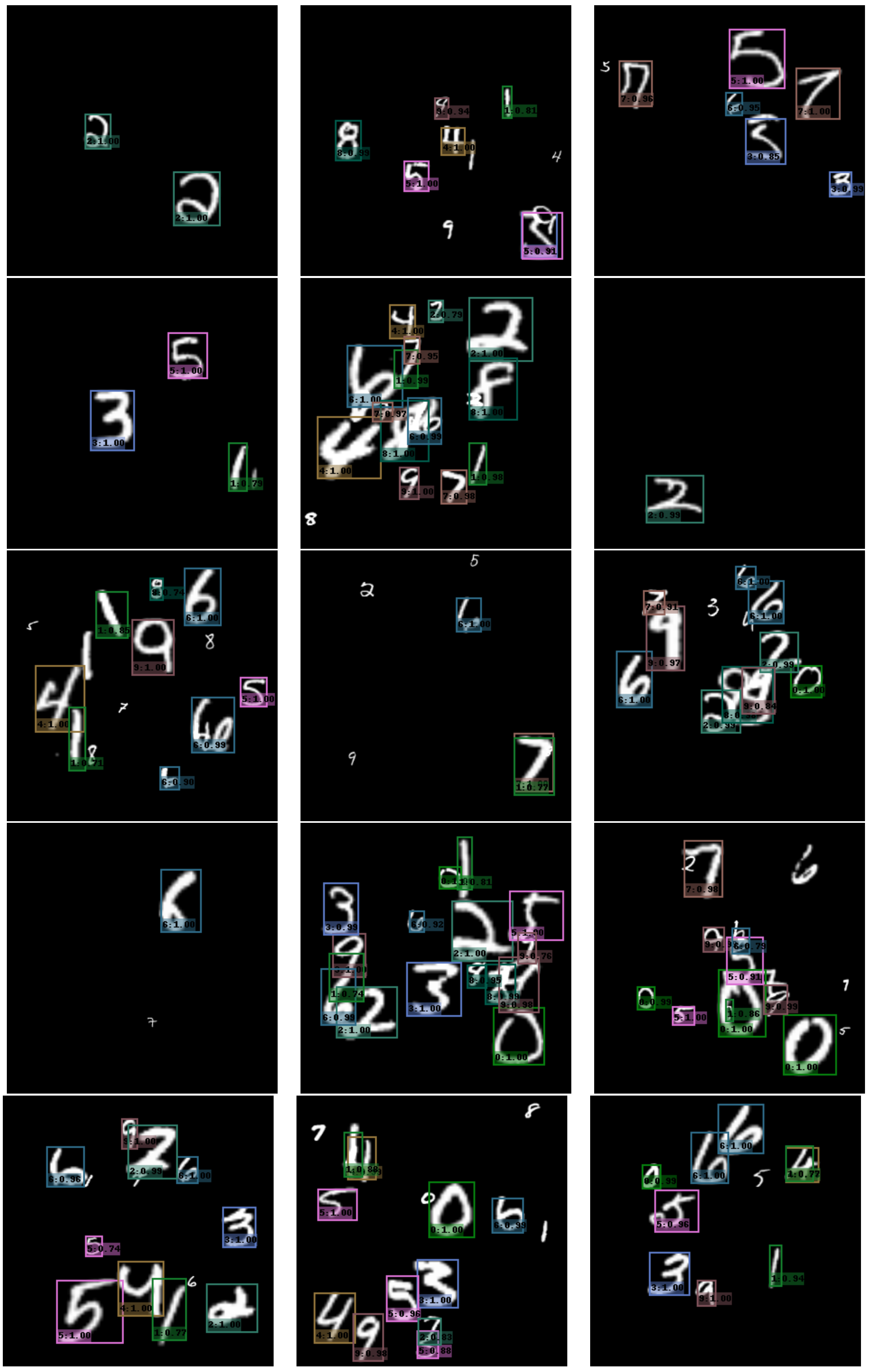


Figure 10: Predictions on the MNIST demoset

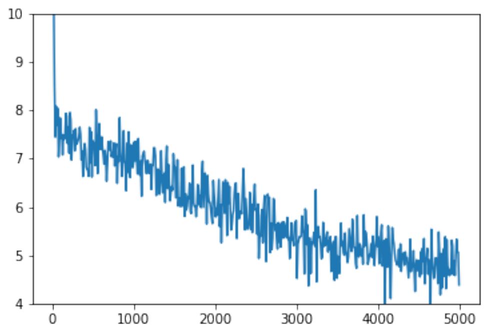


Figure 11: Loss - 5000 iterations.

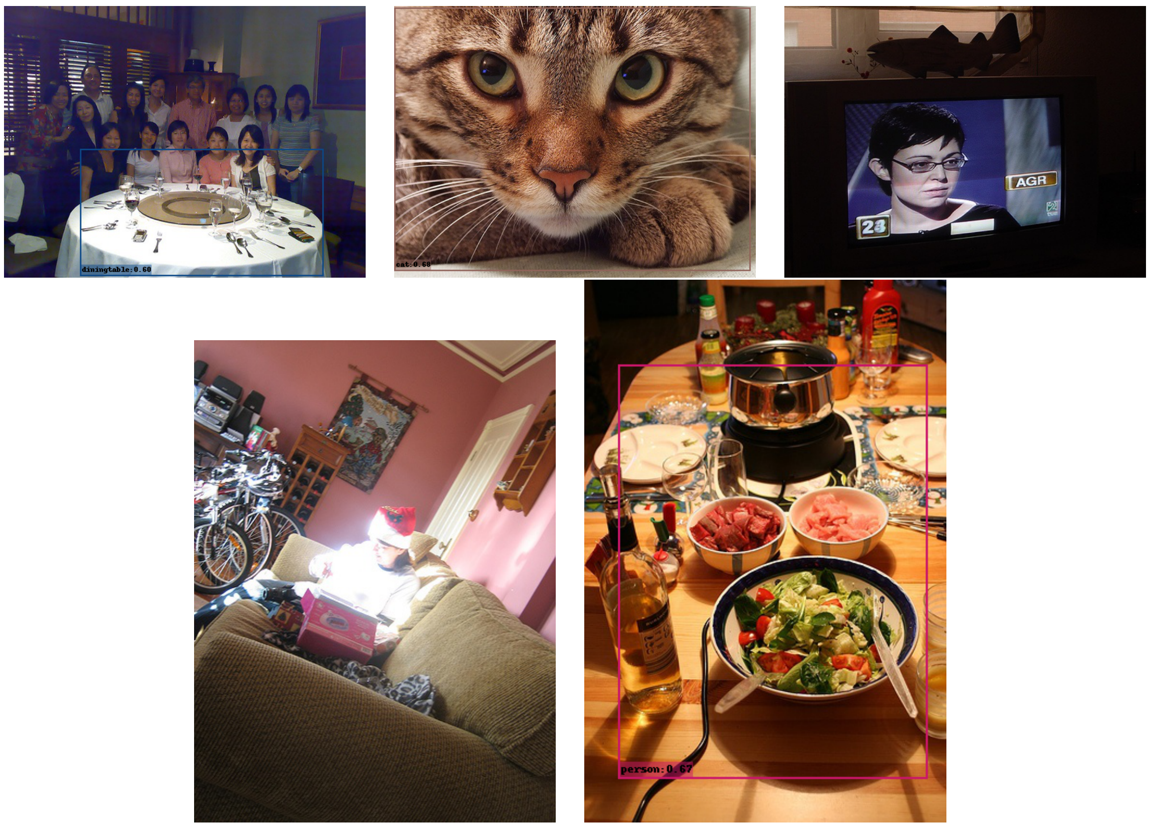


Figure 12: Predictions - threshold = 0.5