Object Detection with Single Shot Multi-Box Detector

Paul Warkentin
p.warkentin@stud.uni-heidelberg.de
University of Heidelberg
Heidelberg Collaboratory for Image Processing (HCI)
Object Recognition and Image Understanding, Prof. Dr. B. Ommer

Motivation

Reasons why I have chosen this specific project:

- Object detection has become more important over the last few years, e.g. in autonomous driving.
- By now, there are many different algorithms for object detection published as papers.
- Just a few are that lightweight to run them on live inference with an appropriate frame rate.
- The Single Shot MultiBox Detector is more efficient on live inference while improving the accuracy at the same time compared to other object detection models like Fast(er) R-CNN and YOLO.

Single Shot MultiBox Detector

What does it stand for?

- **Single Shot:** this means that the tasks of object classification and localization are done in a single forward pass of the network.
- **MultiBox:** this is the name if a technique for bounding box regression developed by *Szegedy et al.*
- **Detector:** the network is an object detector that also classifies those detected objects.

Architecture

- The architecture of the SSD model builds on a classifier architecture.
- Here the VGG-16 architecture was chosen, but others like ResNet or Inception can also be taken as base network.
- The base network is then extended by extra feature layers, thus enabling to extract features at multiple scales and progressively decrease the size of the input to each subsequent layer.

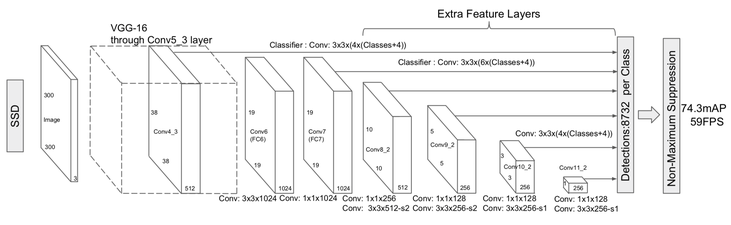
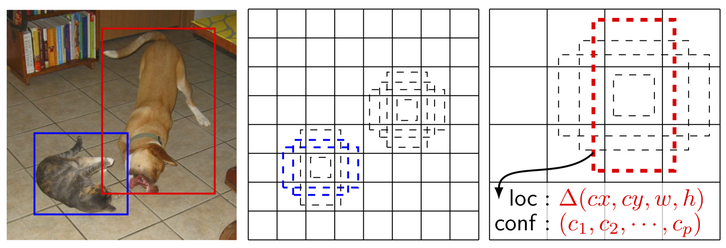


Figure 1: Architecture of the SSD 300 network.

MultiBox Priors

- Default anchor boxes are computed for each feature layer for a given set of aspect ratios and scales.
- The number of anchor boxes for a feature layer of size (h, w) of b different scales and ratios can be calculated by $b \cdot h \cdot w$.
- The SSD 300 in this project has a total of 8732 default anchor boxes.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 2: Matching default anchor boxes.

Loss Function

- Each groundtruth box is matched to the default anchor box with the best Jaccard overlap. Then, default anchor boxes are matched to any groundtruth box with a Jaccard overlap higher than a threshold.
- **Loss:**

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)),$$

where $x_{ij}^p \in \{0, 1\}$ is an indicator for matching the i-th default anchor box to the j-th groundtruth box of class p.

- **Confidence loss:** For each predicted bounding box, a set of class predictions are computed, for every possible class in the dataset.

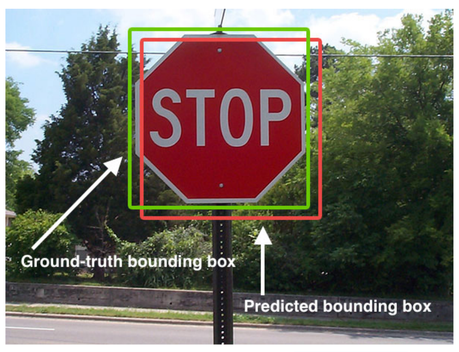
$$L_{conf}(x, c) = -\sum_{i \in Pos} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0),$$

for ever groundtruth box j, where $\hat{c}_i^p$ is the softmax function over the confidences c_i^p.

- **Localization loss:** The localization loss is computed using the smooth L1-Norm.

$$L_{loc}(x, l, g) = \sum_{i \in Pos} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^p \text{smooth}_{L1}(l_i^m - \hat{g}_j^m),$$

where $\hat{g}_j^m$ is the SSD encoding of the groundtruth box j.



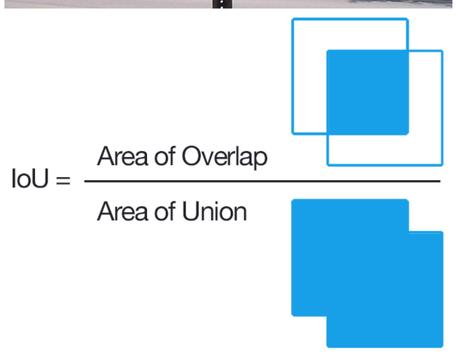


Figure 3: Visualization of the Jaccard score / IoU.

Evaluation

- The metric to evaluate the training results is the mAP (mean average precision).
- Consists of the precision and the recall.
- **Precision:** Measures how accurate is the prediction, i.e. the percentage of the positive predictions are correct.
- **Recall:** Measures how good all the positives were found.

Results

Training

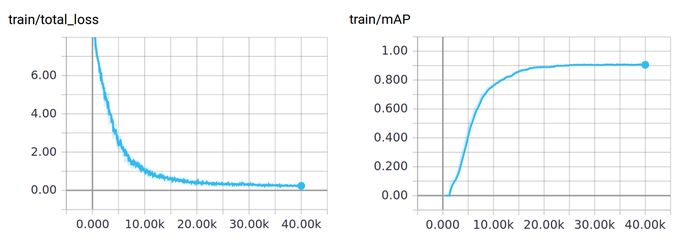


Figure 4: Train loss and mAP evolution of SSD 300 network.

Evaluation

The following results were evaluated on the Pascal VOC 2007 test set.

	Method	Dataset	mAP
	Fast R-CNN	VOC 2007+2012	70.00%
	Faster R-CNN	VOC 2007+2012	76.80%
	SSD 300	VOC 2007+2012	74.30%
	SSD 512	VOC 2007+2012	76.80%
×	SSD 300	VOC 2007+2012	49.09%
×	SSD 512	VOC 2007+2012	54.37%

Table 1: Results compared to the paper.

Inference time

The paper used a Titan X and cuDNN 4 with Intel Xeon E5-2667v3 @ 3.20GHz. The results of this project were tested using a GTX 1080 Ti and cuDNN 9.0 with Intel Core i7-6850K.

	Method	mAP	FPS	BS	# Boxes	Resolution
	Faster R-CNN	73.2%	7	1	~ 6000	$\sim 1000 \times 600$
	Fast YOLO	52.7%	155	1	98	448×448
	YOLO	66.4%	21	1	98	448×448
	SSD 300	74.3%	46	1	8732	300×300
	SSD 512	76.8%	19	1	24564	512×512
	SSD 300	74.3%	59	8	8732	300×300
	SSD 512	76.8%	22	8	24564	512×512
×	SSD 300	49.09%	49	1	8732	300×300
×	SSD 512	54.37%	35	1	24564	512×512

Table 2: Results compared to the paper.

Differences to the paper

- Slightly different data augmentation as in the paper.
- Slightly different pre- and post-processing as in the paper.
- The implementation of the paper was written in Caffe, this projects' implementation was written in TensorFlow.

Sample images

In the samples below, one can see that the model works very good on some images, but often the localization and even the classification of objects is not right. One drawback of the SSD model implemented in this project is that it is not good in detecting small objects. By playing around with the hyperparameters and enough training data, this can be improved.

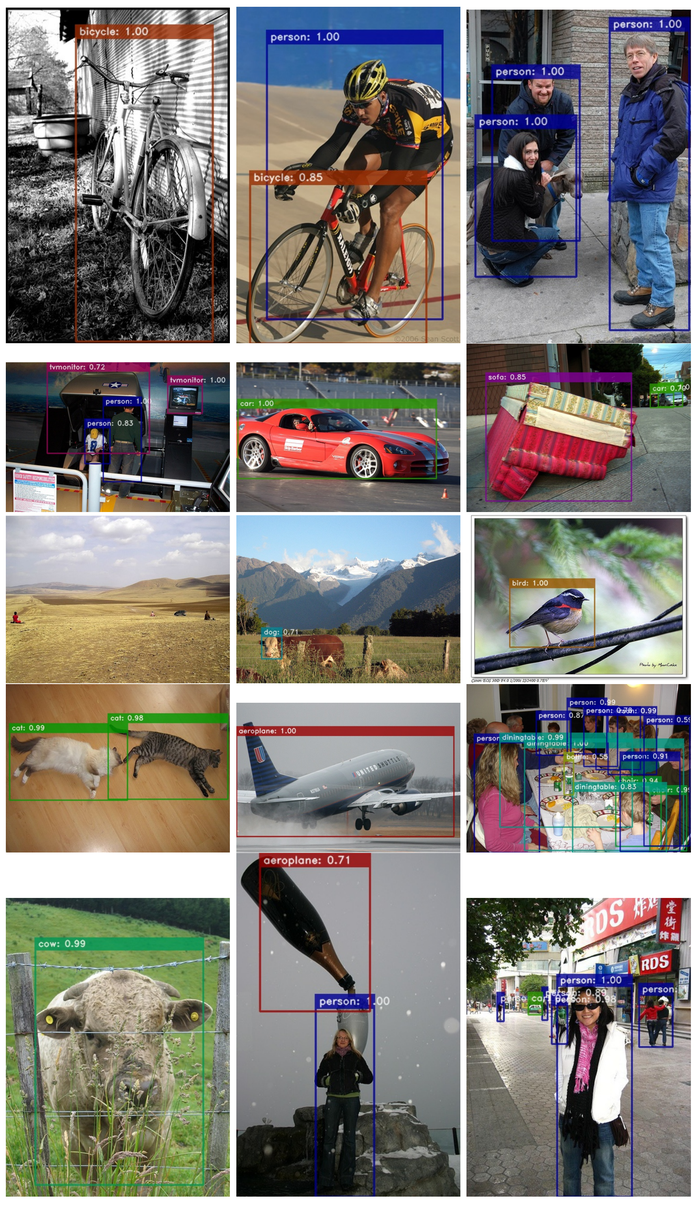


Figure 5: Sample annotated with SSD 300.

Resources

- Original paper about SSD: https://arxiv.org/pdf/1512.02325.pdf