Object Detection: How Noise Patterns Affect the Performance of the Single Shot Multibox Detector

Simon B. Jensen
Software Development & Technology
IT University of Copenhagen
sije@itu.dk

1 Introduction

In the last few years object detection challenges such as the Pascal Visual Object Classes Challenge (Pascal VOC)[1] and the ImageNet Large Scale Visual Recognition Competition (ILSVRC)[2] have seen tremendous increases in detection performances due to new development in Deep Learning methods. More specifically development in Convolutional Neural Networks (CNNs) - a specific branch of Deep Learning. The performance of the top object detectors are now at a point where they have even surpassed human capabilities on these specific challenges. One could, however, argue that the images of these challenges are not realistic as the images are often ideally cropped, non-blurry and mostly only contain a few specific objects, for which the systems are optimized for. The human visual system is seemingly still superior when the settings are more realistic and non-ideal.

We investigate whether this is actually the case, by asking the question: What happens to an object detector's performance if we add noise to the images before feeding them into the detector? Where the reason for adding noise to the images is to simulate a more realistic and non-ideal setting that e.g. also adverse weather scenarios, like rain-, snow- or foggy weather, could impose on a camera sensor. The effects could for instance be a blurred image or an incomplete image etc. Typically humans are able to deviate from these effects and are somehow still able to identify objects when the input is non-ideal.

For this project we investigate how noise influence the detection performance of a state-of-the-art object detection systems, the Single Shot Multibox Detector (SSD) [2]. We use an average filter and salt and pepper noise to simulate adverse weather scenarios. For testing and evaluating the object detection systems we utilize the Pascal VOC data set.

2 The Data: Pascal VOC

The following section introduces and describes the Pascal VOC data set and, in short, it's role as a bench marking data set. The first part of the section can be skipped if the reader is already familiar with the data set.

There exists only a handful of image based data sets which are openly available to the public, and where the images and corresponding labels are both of high quality and there are a substantial amount of them. Thereby containing the key properties for serving as a valid benchmark data set. The Pascal VOC image data set is such a data set. Other such data sets are e.g. The ImageNet, COCO [3] and KITTI Vision [4]. What is significant by publicly available benchmark data sets is the fact that they, in opposition to strictly private or company owned data sets, indulges to good competition between competing factions and can serve as a fair basis for comparison. Competition and the possibility

[1]http://host.robots.ox.ac.uk/pascal/VOC/
[2]http://www.image-net.org/challenges/LSVRC/
[3]http://cocodataset.org/#home
[4]http://www.cvlibs.net/datasets/kitti/

for comparison will typically motivate and provoke even faster development in an area. And this is exactly what the Pascal VOC data set has done, since the first Pascal VOC challenge started in 2005, for the area of object detection among other computer vision challenges. It has since been held as a yearly competition, each year presenting new systems which pushed the boundaries for what is possible for computer vision systems. In this project we utilize the Pascal VOC data set for evaluation of the SSD object detection system. It should be noted that the Pascal VOC competition was last time held in 2012. But the evaluation server is still online and the data set can easily serve as a valid basis for the purpose of this project.

2.1 Pascal VOC and Object Detection

In this project we focus on the object detection part of the Pascal VOC data set, as we are interested in testing the robustness of an object detector. But the Pascal VOC competition also consists of numerous other computer vision challenges, such as image classification, instance segmentation and more advanced challenges such as action classification.

The object detection part of the challenge consists of a training-, validation- and test data set which consists of images with at least one, but possibly multiple objects in them. There are 20 different object classes which are shown, together with their quantities in the train-, validation- and test set, in table 1). As can be seen, the distribution of object classes in the data set is very uneven meaning that some object classes are much more frequent than others. The number of images in the data set is distributed as follows:

- Number of training images (Pascal VOC 2007 + Pascal VOC 2012): 8218
- Number of validation images (Pascal VOC 2007 + Pascal VOC 2012): 8333
- Number of test images (Pascal VOC 2007): 4952

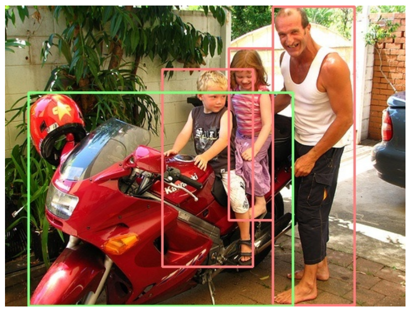
The goal of the Pascal VOC object detection challenge is to detect both the correct object classes and locations of potentially multiple objects in an image. It should be noted that multiple objects from multiple classes may be present in the same image. The location of an object is simply represented by a 2D bounding box which surrounds the extent of the object visible in the image. When determining how well a object detector is performing the evaluation metric used is the mean Average Precision (mAP), which we explain in section 5. The Pascal VOC images typically depict realistic scenes (see figure 1 for examples). The data set consists of images crawled from flickr and from Microsoft Research Cambridge (MSRC) data set. For more information on the data set we refer to `http://host.robots.ox.ac.uk/pascal/VOC/#history`.

Object class	**Train (#)**	**Val (#)**	**Test (#)**	**Total (#)**
aeroplane	626	659	311	1596
bicycle	612	596	389	1597
bird	886	934	576	2396
boat	716	681	393	1790
bottle	1087	1029	657	2773
bus	448	461	254	1163
Car	2017	1991	1541	5549
cat	800	816	370	1986
chair	2183	2155	1374	5712
cow	540	518	329	1387

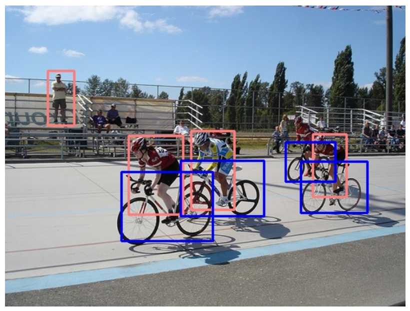
Object class	**Train (#)**	**Val (#)**	**Test (#)**	**Total (#)**
diningtable	521	536	299	1356
dog	1039	1040	530	2609
horse	584	572	395	1551
motorbike	568	573	369	1510
Pedestrian	7724	7852	5227	20803
pottedplant	862	862	592	2316
sheep	700	647	311	1658
sofa	617	594	396	1607
train	485	499	302	1286
tvmonitor	603	590	361	1554

Table 1: *The 20 Object classes of the Pascal VOC data set and their quantities in the 2007 + 2012 train- and validation data sets and the 2007 test data sets*

For this project we test the object detection system on both the original 4952 unmodified Pascal VOC test images, but also on the test images which have been blurred or which have added salt and pepper noise, as described in section 3.2. Thereby gaining knowledge about how robust the object detector is toward noise, which is suppose to simulate the effects adverse weather scenarios could impose on a



(a) *Image of three Person objects and a Motorbike object*



(b) *Image of four Bicycle objects and four Person objects*

Figure 1: *Images from the Pascal VOC test data set. 2D Bounding Boxes indicate the location and object class of the objects according to the ground-truth labels*

camera sensor. We use pre-trained weights which have been trained using the entire combined Pascal VOC 2007 + 2012 training- and validation datasets but this is further described in subsection 4.3.

3 Noise Patterns and Object Detection

The following section argues why it is relevant to test an object detection systems on input images with added noise and further introduces the two noise patterns used in this project, blur by an average filter and salt and pepper noise.

3.1 Motivation: Why does Noise on Images matter?

Let us dwell upon the question of why we are interested in testing how object detection systems perform on images with added noise? Let us consider the case of a vehicle driving on a rainy day. The rainy weather makes the outside world look different, when seen through the car windows, from how it usually looks on a clear day. The traffic lights, the other cars in the traffic and the pedestrians has possibly become more blurry and unclear due to the water on the car window.

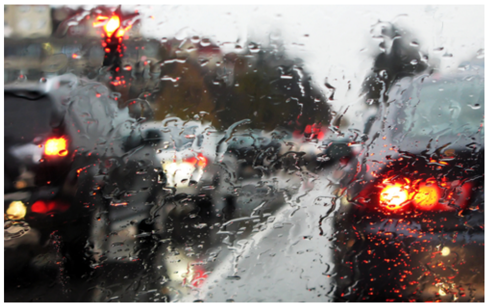
When a human driver is sitting behind the steering wheel the rainy weather means it becomes harder for the driver to detect objects he or she sees through the car windows. But the human visual perception system is nevertheless able to deviate well enough from these changes so that we can navigate (more or less) safely in the traffic, even through adverse weather scenarios. Other weather circumstances could likewise change the looks of the world seen through the car windows. Consider snowy weather or foggy weather. Figure 2a and 2b show examples of how rain and snow can affect the world seen from inside the car.

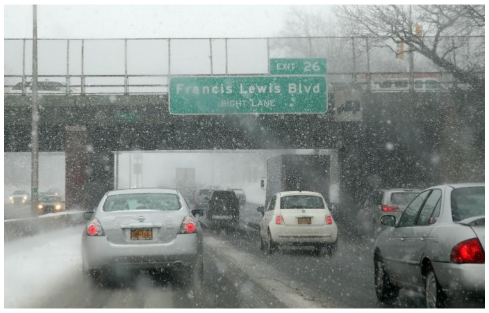
3.2 Average Filtering Blur and Salt and Pepper Noise

In order to simulate different real world adverse weather scenarios we apply blur by using an average filter and salt and pepper noise to the images. The blur is meant to simulate the impact rainy- or foggy weather could have on an camera sensor capturing input images. The salt and pepper noise is equally meant to simulate the impact rainy- or snowy weather could have on a camera sensor. We use kernels of size 7x7 and 9x9 for the average filtering blur. The effect of the blur can be seen in figure 3b and 3c. The average filtering blur is simply applied by sliding a kernel of a specific size (7x7 or 9x9 in our case) over the image pixels and averaging the pixel values.

We implemented the following simple Python function for applying the average filter blur, which requires the OpenCV and Numpy packages:

```python
def average_filtering_blur(input_image, kernel_size):
    kernel = numpy.ones(shape=(kernel_size, kernel_size), dtype=np.float32)
    avg_kernel = kernel / (kernel_size ** 2)
    blurry_image = cv2.filter2D(src=input_image, ddepth=-1, kernel=avg_kernel)
    return blurry_image
```


(a) *Image of the possible effects of rain weather*



(b) *Image of the possible effects of snow weather*

Figure 2: *Examples of the possible effects adverse weather scenarios could impose on the vision from the inside of a car. A camera sensor, capturing input images to an object detector, could likely experience the same effects. Images were found by a simple Google search. Original image urls are:* `http://www.carguide.ph/2016/08/driving-in-downpour-practical-rainy-day.html` *and* `http://www.nydailynews.com/news/national/snor-easter-blizzard-spares-new-york-city-article-1.2093191`

For the salt and pepper noise we implemented the following Python function which equally requires the Numpy package:

```python
def salt_and_pepper(input_image, salt_and_pepper_amount):
    salt_vs_pepper = 0.5    # 50% Salt/50% Pepper

    # Salt mode
    n_salt = np.ceil(salt_and_pepper_amount * input_image.size * salt_vs_pepper)
    random_salt_coordinates = [np.random.randint(0, i-1, int(n_salt)) for i in img.shape]
    salty_image = input_image[random_salt_coordinates] = 1

    # Pepper mode
    n_pepper = np.ceil(salt_and_pepper_amount * input_image.size * (1. - salt_vs_pepper))
    random_pepper_coordinates = [np.random.randint(0, i-1, int(n_pepper)) for i in img.shape]
    salt_and_pepper_image = salty_image[random_pepper_coordinates] = 0

    return salt_and_pepper_image
```

The variable, **salt_and_pepper_amount**, is set to 0.064 for the first salt and pepper experiment and to 0.128 in the second experiment. The effect of the salt and pepper noise can be seen in figure 3d and 3e. As mentioned, the noise should simulate the effect rainy-, snowy- or other adverse weather types could impose on a camera sensor. A comparison between the images in figure 2 and the images in figure 3 show that this assumption is not entirely far-fetched.

4 The Object Detector: Single Shot Multibox Detector (SSD)

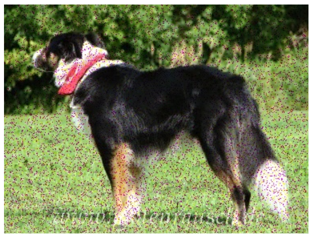
The following section introduces and describes the CNN based object detection system, SSD [2], which is used and tested during the project. The system was, as of 2016, a state-of-the-art object detection systems, showing detection speeds faster than the YOLO network [3] while still obtaining prediction accuracy close to Faster R-CNN [4]. Still today, SSD and it's many derivatives are among the top performing object detectors. For this project we utilize a TensorFlow implementation of the SSD-300 network [5]. In table 2, the ranking of the original SSD-300 and SSD-512 implementations on the Pascal VOC 2012 evaluation server can be seen. The distinction between SSD-300 and SSD-512 lies only on the size which the input image is scaled down/up to before it is fed to the network. Naturally, the SSD-300 is therefore a more lightweight edition of the SSD network than the SSD-512 and it is therefore also showing a slightly lower average prediction accuracy.

[5]https://github.com/balancap/SSD-Tensorflow

(a) *The original untouched image*

(b) *Image blurred by a 7x7 average filter*

(c) *Image blurred by a 9x9 average filter*


(d) *Image with added salt and pepper noise*

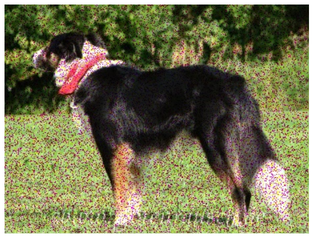

(e) *Image with even more added salt and pepper noise*

Figure 3: *Image (000018.jpg) of a dog from the Pascal VOC 2007 test set*

Ranking	**Name**	**mAP (%)**	**max AP (%)**	**min AP (%)**	**submission date**
16	SSD-512	82.2	93.9 (cat)	60.9 (potted plant)	10-Oct-2016
27	SSD-300	79.3	93.4 (cat)	50.2 (potted plant)	18-Oct-2016

Table 2: *The original SSD-300 and SSD-512 implementations' ranking on the Pascal VOC 2012 evaluation server . SSD-300 and SSD-512 is ranking 27 and 16 respectively (as of 07.12.2017) with a mean average precision (mAP) of 79.3% and 82.2%. The results are found at:* `http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=4`

4.1 The Backbone: VGG16

Before going deeper into the SSD implementation it is worth first understanding it's **backbone**. By backbone we mean the CNN, which the images are fed into and which output good and useful feature maps for the object detection system to further work on. We assume that the reader is already familiar with CNNs and concepts such as convolutional- and pooling layers and the Rectified Linear Unit (ReLU) activation function. Most of today's successful object detection systems include one of the following popular standard CNNs in their architecture: The VGG-16 or VGG-19 (from University of Oxford)[5], Google's Inception models[6] or Microsoft's ResNets[7]. Though the architectures of these CNNs are somewhat different they fundamentally do the same and they show only marginal difference in performance on classification tasks. What is great about them is the fact that they have been proven to have great classification performance on the ImageNet data set, thereby guaranteeing that the feature maps they output are of good quality, and it is possible to find and download pre-trained versions of them, which can save many days of training time. In this project we use the VGG-16 CNN as backbone, like the original SSD-300 implementation.

What do these backbone CNNs then do that is so useful? As we mentioned the CNNs take images as input and output feature maps for the object detector to work with. The feature maps are maps of specific object features which the CNN has extracted from a given input image. The object features are extracted by passing the input image through a series of convolutional-, pooling- and activation layers. Where the weights of the convolutional layers have been learned during training. So basically the CNN, during training, have learned which features to look for in an input image and the feature maps are then the features which the CNN have extracted for a specific input image.

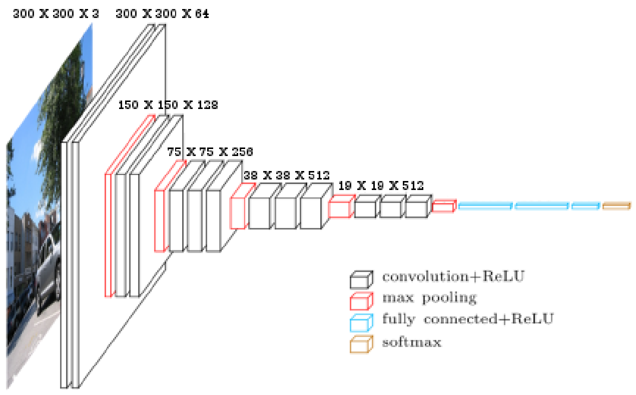


Figure 4: *The original VGG-16 architecture (with dimensions modified for SSD-300). The last pooling layer, the three fully connected layers and the softmax layer are all excluded when VGG-16 is used as backbone CNN in the SSD network. The image originates from the original VGG paper [5]*

For the original VGG-16 implementation the number of convolutional layers, which parameters are learned during training, are 13. The number 16 comes from the fact that VGG-16 actually includes 3 fully connected layers at it's end when it is used in classification tasks. However, the 3 fully connected layers are excluded when VGG is used as a backbone in the SSD network. Figure 4 visualizes the VGG-16 architecture.

4.2 Moving from Classification to Object Detection

Now how do we move on from the SSD backbone, the VGG-16 CNN, which is able to perform classification tasks, to being able to perform object detection? What makes object detection such a complex task compared to classification is the fact that an image potentially contains multiple objects of interest, and for each of the objects both the class and localization in the image need to be determined.

One of the ways SSD tackles this problem is by adding several feature layers of different scales to the end of the feature maps outputted by the VGG-16 CNN. For the SSD-300 model the number of added feature layers is 7 as can be seen in figure 5 (one is actually inside the VGG-16 body).

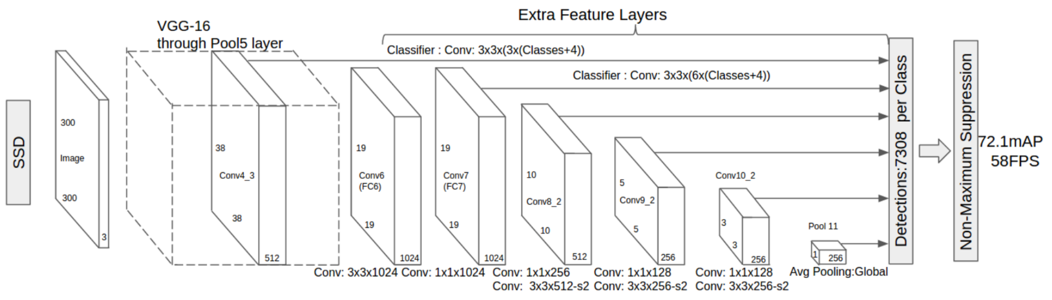


Figure 5: *The original SSD architecture. The image originates from the original SSD paper [2]*

The reason for adding feature layers of different scales is to be able to detect objects of many different sizes as the receptive fields, with respect to the input image, of each feature layer will be different.

The feature layers are then discretized into a set of so called default boxes (equivalent to R-CNN's anchor boxes) of different aspect ratios and sizes and tried matched onto the feature layers. This procedure is best visualized with an image as can be seen in figure 6

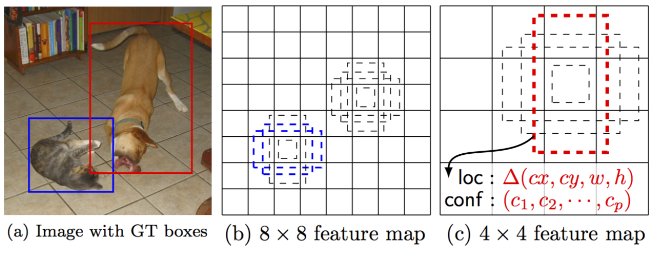

loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$
(a) Image with GT boxes
(b) 8×8 feature map
(c) 4×4 feature map

Figure 6: *The SSD evaluates a small set of default boxes of different aspect ratios at each location in several feature layers with different scales (e.g. 8x8 and 4x4 as seen in (b) and (c))*

This approach to object detection has allowed actual real-time detection speeds. We refer to [2] for further information on the approach and on the model loss function, which is a weighted sum between localization loss and confidence loss.

4.3 SSD in TensorFlow

We use a TensorFlow implementation of the original SSD-300 network which can be found at `https://github.com/balancap/SSD-Tensorflow` and which has shown detection results equivalent to the original SSD implementation on the Pascal VOC 2007 test set. The implementation includes downloadable pre-trained weights for the SSD-300 model, trained on the entire Pascal VOC 2007 and 2012 training- and validation sets. The model works with data in the TFRecords file format[6], so in order to evaluate the SSD model, it is necessary to first convert the Pascal VOC datasets into TFRecords. We do that using the repository's included conversion script "tf_convert_data.py" and run the following command:

```
$ python tf_convert_data.py \
$ --dataset_name=pascalvoc \
$ --dataset_dir=[PATH to Pascal VOC]/ \
$ --output_name=voc_2007_test \
$ --output_dir=[PATH to store TFRecords]/
```

In order to evaluate the model we use the repository's evaluation script: "eval_ssd_network.py" and run the following command:

```
$ python eval_ssd_network.py \
$ --eval_dir=[PATH to store evaluation results and logs] \
$ --dataset_dir=[PATH to Pascal VOC TFRecords]/ \
$ --dataset_name=pascalvoc_2007 \
$ --dataset_split_name=test \
$ --model_name=ssd_300_vgg
$ --checkpoint_path=[PATH to pretrained weights]/ssd_300_vgg.ckpt \
$ --batch_size=1
```

We repeat this process for the Pascal VOC 2007 test data which has been blurred by an average filter as well as with added salt and pepper Noise and obtain the results shown in table 3.

What we see in table 3 is that the SSD-300 TensorFlow implementation is achieving a mean average precision of 82.1 % on the original unmodified Pascal VOC 2007 test data. This is a surprisingly high number, given that the model, according to the repository documentation, has only been trained

[6]The TFRecords file format is a simple record-oriented binary format that many TensorFlow applications use for training data

Input	mAP (%)	max AP (%)	min AP (%)
Original Images	82.1	90.9 (cat)	58.2 (bottle)
Avg. Filter (7x7)	66.6	84.4 (cat)	36.5 (bottle)
Avg. Filter (9x9)	59.1	78.1 (cat)	28.7 (bottle)
S&P (0.064)	71.9	88.0 (train)	46.6 (bottle)
S&P (0.128)	65.8	81.2 (horse)	34.9 (boat)

Table 3: *The performance of the SSD-300 TensorFlow implementation on the original untouched Pascal VOC 2007 test data set and on the Pascal VOC 2007 test data set modified by blur and salt and pepper noise*

on the Pascal VOC 2007 + 2012 training and validation data set. The performance even exceeds the performance of the original SSD-300 implementation. What we also see in the table is that the mean average precision, when an average filter with a kernel size of 9x9 is applied to test images, decrease from 82.1% to 59.1%. This is a relative decrease of 28%. In the case where salt and pepper noise is added to the test images, with the salt and pepper amount variable set to: 0.128, the mean average precision decreases from 82.1% to 65%, which corresponds to a relative decrease of 20%. The entire evaluation results for each individual class is found in appendix A in table 5. The relative changes of each class can as well be found in the appendix in table 6. An interesting observation to note is that the two object classes, bus and train, have some of the lowest relative changes, even though the classes are among the quantitatively sparsest. A reason for this, could be the fact that both classes have very distinct features, e.g. the shape of the bus and train body. The pedestrian object class accounts for approximately one third of all the objects in the Pascal VOC data set but the decrease in performance is as high as for the bus and train object classes. The object classes with the biggest relative decreases are the bottle and boat classes, which deceases with 49.9% and 50%, respectively when the average filter blur (with kernel size 9x9) is applied to the test images. Another interesting observation here is to see that e.g. the average precision of the bottle class decreases much more drastically when blur is applied to the test images than when salt and pepper noise is added.

5 Precision/Recall and Mean Average Precision

A common metric used for evaluating object detection performance is the mean average precision (mAP). In order to understand the metric one first need to understand precision and recall. Precision and recall are given by the following equations:

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

Where TP stands for **true positive**, FP stands for **false positive** and FN stands for **false negative**. A true positive, with respect to object detection, is the case where a predicted bounding box has an overlapping area with the ground-truth bounding box above a certain threshold value. The overlapping area is measured as the intersection over union (IoU):

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}$$

For the Pascal VOC the IoU value has to be above 50% for the predicted bounding box to count as a valid true positive. Figure 7 shows an illustration of overlapping bounding boxes. The threshold value of 50% is intentionally set this low, by the people behind Pascal VOC, to account for possible inaccuracies in the ground-truth bounding boxes. In a real world scenario where e.g. a vehicle navigates in the traffic the threshold value should likely be a lot higher, as not knowing the exact position of an object could potentially jeopardize lives.

A false positive, with respect to object detection, is when a non-object like e.g. the foreground/background or an object of another object class is predicted as being a specific object class. When evaluating the performance of Pascal VOC multiple predictions of the same object also count as false positives. If for instance the same object is predicted by 4 different bounding boxes only the first will count as a true positive the rest will count as false positives. Lastly, a false negative, with respect to object detection, is when no bounding box is predicted for the object of interest.

There exists a inverse relationship (a trade-off) between the precision and recall values. So when one of them increases the other decreases. A precision/recall curve for each object class can be computed by changing a threshold value for which the detector dares to make predictions. The average precision is found by finding the value on the precision/recall curve where the sum of the precision and recall values are maximized. The mean average precision is simply taking the mean of the average precision found for each object class.

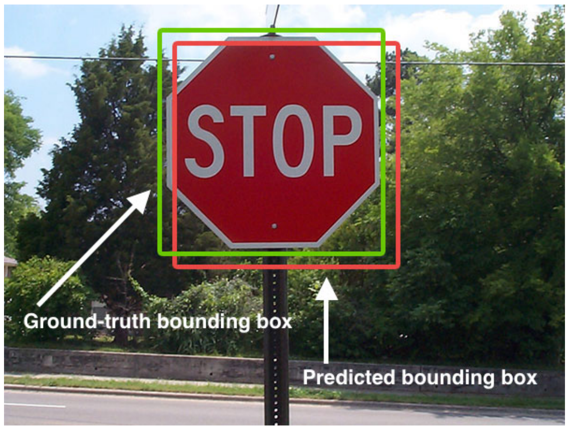


Figure 7: *A predicted bounding box overlapping with a ground-truth bounding box. The image can be found at url:* `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`

6 Perspectivation

The result shown in table 3 shows that, when blur and salt and pepper noise is added to the Pascal VOC test images, the performance (measured in the mAP) of the SSD object detector decreases. This could also be the case for other object detecting systems. So it is worth asking, how we can deal with the challenges which the noise patterns impose on SSD and likely other object detectors. The following subsections present three approaches which could possibly enhance the robustness of the object detectors.

6.1 Generative Adversarial Neural Network

One way that could possible increase the robustness of the object detector, towards noise, could be to pass input images through a Generative Adversarial Neural Network (GAN) before feeding them to a object detection systems. GANs are a special branch of artificial neural networks, which have been proven to be very effective when it comes to de-noising signals[1]

6.2 Image Augmentation

Another possible approach could be to utilize the concept of image augmentation. Image augmentation is, like the name suggests, a way to augment the image data, such that it become more useful for a model performing a computer vision task. A data augmentation method, which SSD actually already utilizes, is for instance to extend the training set with copies or the original training data set, but adding color saturation to the copies. Another data augmentation method which SSD already utilizes is to extend the training data with images which zooms in on the objects marked by the ground-truth labels. There are a million other ways the training images could be augmented which could possibly increase the robustness of the object detector.

6.3 Sensor Fusion

Another possible approach, which could make sense when the input data is served to the object detector by a camera sensor, is to increase the number of input sensors the model relies on. And further utilize fusion methods, such that if one sensor fails to deliver a good input image the model will rely on another sensor. In relation to this, a possible approach could be to extend the model to rely on a fused signal from multiple different types of sensors. e.g LIDAR data, radar, GPS and/or gated cameras.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Curran Associates, Inc.*, 2014.

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[3] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[4] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[7] Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron C. Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *CoRR*, abs/1505.00393, 2015.

Appendices

A SSD-300's Mean Average Precision on Pascal VOC

Input	**mAP**	**aeroplane**	**bicycle**	**bird**	**boat**	**bottle**	**bus**	**Car**	**cat**	**chair**	**cow**	**diningtable**	**dog**	**horse**	**motorbike**	**Pedestrian**	**pottedplant**	**sheep**	**sofa**	**train**	**tvmonitor**
Original	82.1	88.3	83.3	80.7	72.7	58.2	88.7	84.8	90.9	68.4	87.1	78.8	90.7	89.8	87.8	82.8	67.1	80.1	86.7	90.3	85
7x7	66.6	71	72.4	59.2	46.6	36.5	74.8	68.4	84.4	47.2	68.4	65.8	81.9	78.8	77.6	70	46	60.5	73.8	82.6	66.2
9x9	59.1	62.2	65.4	49.9	36.4	28.7	70.1	62.6	78.1	38.6	57.5	59.1	73.4	73.8	72	64.6	38.5	51.4	66.4	74.2	59.7
S&P 1	71.9	62.3	78.9	64.9	47.3	46.6	82.2	76.6	85.7	58.8	77.8	73.1	82.9	84.7	81.4	77.1	54.3	64.8	76.5	88	73.8
S&P 2	65.8	53.6	76.2	54.1	34.9	41.9	76.1	70.6	80	53.7	70.1	69.7	77.5	81.2	77.2	73.1	49.5	60.4	69.8	79.9	67.5

Table 4: The average precision of each class when evaluating the TensorFlow implementation of the Single Shot Multibox Detetor (SSD-300) on the original unmodified Pascal VOC test data set, on the Pascal VOC test data with added blur (using average filtering with different kernel sizes) and on the Pascal VOC test data with added salt and pepper noise

Input	**mAP**	**aeroplane**	**bicycle**	**bird**	**boat**	**bottle**	**bus**	**Car**	**cat**	**chair**	**cow**	**diningtable**	**dog**	**horse**	**motorbike**	**Pedestrian**	**pottedplant**	**sheep**	**sofa**	**train**	**tvmonitor**
7x7	-18.9	-19.6	-13.1	-26.7	-35.9	-37.2	-15.7	-19.3	-7.2	-31	-21.5	-16.5	-9.7	-12.3	-11.7	-15.5	-31.4	-24.5	-14.9	-8.6	-22.1
9x9	-28	-29.5	-21.5	-38.2	-49.9	-50.7	-21	-26.2	-14.1	-43.6	-34	-25	-19	-17.8	-18	-22	-42.6	-35.8	-23.4	-17.9	-29.8
S&P 1	-12.5	-29.4	-5.3	-19.6	-34.9	-19.8	-7.4	-9.6	-5.7	-13.9	-10.7	-7.2	-8.6	-5.7	-7.4	-6.9	-19.1	-19.2	-11.8	-2.6	-13.2
S&P 2	-19.8	-39.3	-8.5	-33	-52	-27.9	-14.3	-16.7	-12	-21.5	-19.6	-11.5	-14.5	-9.6	-12.1	-11.7	-26.1	-24.6	-19.5	-11.6	-20.6

Table 5: The table shows the relative change of the mean average precision obtained by the SSD-300 TensorFlow implementation, when noise is added to the Pascal VOC test images, compared to mean average precision obtained on the unmodified test images. In all cases the relative change is, not surprisingly, negative, which is marked with a negation sign