MASTER THESIS

Erasmus School of Economics

Econometrics and Management Science: Business Analytics and Quantitative Marketing

White-Boxing Object Detection with Occlusion Experiments

Author:
Martijn Cazemier
411884

Supervisor:
Ş. Ilker Birbil

Second Assessor:
Michel van de Velden

Abstract

Interpretability is an important property of many machine learning and statistical methods. However, deep learning based methods, including those for object detection, have a black box like character and lack this valuable property. We apply occlusion experiments in a new context and show how they can be used to increase the interpretability of object detection methods. The results of the occlusion experiments are visualized with heat maps, revealing which image patches are used by the detection methods during inference. Through applications with a Faster R-CNN and a single shot detection model, we show that conducting occlusion experiments can help answer questions such as: (i) What information does the model use to decide where to draw the bounds of the box? It turns out that the edges of the object are not always important. (ii) How does the context around the object influence the model's ability to accurately detect the object? We observe that non-informative context can prohibit detection. (iii) Which parts of the object does the model recognize? When part of an object has an unusual shape, we observe that the models might be able to identify this unusually shaped segment.

June 7, 2019

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I want to thank my thesis supervisor, Ilker, for being supportive throughout the research and writing process. You were primarily positive and enthusiastic but also critical at the right time, pushing me to rethink my ideas, which helped me to continuously redirect my research into a fruitful direction.

I am grateful for the opportunities and experiences that the Erasmus University Rotterdam has given me over the past years. Moreover, I am grateful for the the wealth of knowledge that the staff of the Econometric Institute of the Erasmus School of Economics has enriched me with. In particular, gratitude goes to my second assessor, Michel van de Velden, who has done an excellent job introducing me to the field of machine learning and inspired me to pursue a thesis topic in this field.

An important role has been played throughout the entire research project by the active online machine learning community, including seasoned researchers sharing their findings, generous open source developers sharing their code, maintainers and users of question and answer platforms who have already formulated and answered my questions upfront, and companies such as Google and NVIDIA. These two companies have provided me with a free cloud computing service called Google Colaboratory, which I have used extensively, and Google has build the deep learning framework TensorFlow as well as an object detection API, which I have greatly benefited from.

Next, I want to thank my girlfriend, Nora, who often patiently listened while I would be thinking out loud to her. Throughout the past months I have been introducing her to the field of machine learning to the point where she started to proof-read my writing for a lot more than just a check of the spelling and the grammar.

Finally, I want to thank my parents, Eveline and Rogier, my close family, and my friends who are always there for me, give me confidence, give me energy, and bring me joy.

1 Introduction

Current state of the art methods for most computer vision tasks have convolutional layers at their core, and the success of these methods depends on those layers. Although convolutional neural networks (CNNs) are widely used, the inner processes of the networks remain poorly understood. The black box like character of deep neural networks (DNNs) in general, and CNNs in particular, is commonly criticized upon. It is desirable to be able to interpret how a deep CNN has come to a certain prediction or classification decision. Namely, such interpretations can help researchers improve the networks and can be valuable in themselves when a model is deployed in practice.

The current research addresses the problem that CNNs lack interpretability. More specifically we focus on CNN based object detection methods. Object detection methods are neural networks that are supposed to identify each instance of a prespecified set of objects in an image. Such models identify an object by outputting the coordinates of a rectangular box which is supposed to tightly enclose the object. Additionally, a classification score is output indicating how confident the model is that the object belongs to a certain class. The two object detection methods under consideration are called Faster R-CNN and *Single Shot MultiBox Detector* (SSD). Instances of these methods are trained and tuned to detect handguns. We study these trained models in order to find out how the models make use of the available information in the input images. Correspondingly, the research question of this thesis reads: *How do deep learning based object detection methods make use of the available information in an unseen image?*

In order to find answers we use a technique called the occlusion experiment. The occlusion experiment is one of several visualization techniques introduced by Zeiler and Fergus (2014) for the purpose of interpreting image classification methods. Their work represents a substantial contribution related to the research topic of white-boxing CNNs. The current paper applies the occlusion experiment in a *new* context.

An occlusion experiment is conducted as follows: A grey occlusion box slides over an image and the models repeatedly perform inference. For each position of the box, the localization accuracy and the classification confidence corresponding to the correct class are recorded. So, the effect of occluding different parts of the image is measured. Subsequently, this effect can be visualized using a heat map. We use these heat maps to observe which information the model uses to detect the object. The heat map serves us well because the effect of the occlusion (i.e. the darkness of the heat map) relates to the degree to which the inference depends on the information contained by the occluded image patch.

1.1 Literature Review

The field of computer vision has been progressing quickly over the past decade. Especially since the breakthrough paper by Krizhevsky et al. (2012), who describe the first successful application

of convolutional neural networks (CNNs) to the task of image classification. This task comprises of predicting a single class for a given image. Since its initial succes, CNNs have proven to be useful for other computer vision tasks as well, including image captioning, semantic segmentation, and object detection. The latter computer vision task is of interest to us here and recent methodological developments pertaining to this task are briefly discussed in this subsection. Additionally, two influential related works on interpreting convolutional neural networks are mentioned and an article on pistol detection is discussed.

Girshick et al. (2014) were the first to develop an object detection method that successfully takes advantage of the strengths of convolutional layers. The method Girshick et al. (2014) introduced is called Region based Convolutional Neural Network (R-CNN), which drastically outperformed all other object detection methods at the time. Later, Girshick (2015) developed the Fast R-CNN model, which gains detection speed over R-CNN by sharing computation and requiring only a single forward pass through the convolutional layers of the network. The region proposal based object detection method Faster R-CNN (Ren et al., 2015) is the successor of Fast R-CNN and R-CNN. Faster R-CNN uses a region proposal network, which eliminates the bottleneck that arises from using a selective search type algorithm. This enables Faster R-CNN to yield near real-time object detections, reportedly at a rate of about five frames per second (fps). Even faster models have been developed. Redmon et al. (2016) have invented a model called *You Only Look Once* (YOLO) and Liu et al. (2016) have proposed the *Single Shot MultiBox Detector* (SSD) model. Of these two models, SSD yields the best prediction accuracy.

Zeiler and Fergus (2014) came up with the idea of doing occlusion experiments to find out what information is used by a convolutional neural network. In their work they also discuss a way to visualize high level features. The neurons higher up in the network learn to be responsive to specific patterns in the input. Visualizing high level features entails visualizing the pattern in the image that caused a certain neuron to activate and this is what Zeiler and Fergus (2014) managed to do. Simonyan et al. (2013) introduce the concept of saliency maps, which can be used to find out what information an image classification model uses. A saliency map is obtained by computing the gradient of a class score with respect to the input image.

The models that are subject to an interpretative analysis in the current thesis are pistol detection models. Olmos et al. (2018) have also focused on handgun detection because they recognized an opportunity to apply CNNs in the context of law enforcement. Namely, they develop a Faster R-CNN pistol detector that can be integrated into an automatic alarm system, which could improve public safety. However, when it comes to realizing such an alarm system, lack of interpretability represents a hurdle. Namely, at the current state of technology, these type of models are prone to making mistakes. It is neither straightforward to find out what causes a deep learning based object detection model to make a mistake nor is it straightforward to find out what information a correct detection is based on. Without the functional understanding of its strengths and weaknesses, relying on an automatic alarm system powered by deep CNNs is too risky. So, lack of

interpretability is a serious obstacle in this case.

1.2 Contributions

This thesis contributes to the existing academic literature in the following ways: To the best of our knowledge, this is the first research to apply occlusion experiments (Zeiler & Fergus, 2014) in the context of object detection. By doing so, we show that this technique is a valuable tool for interpreting the inferences made by the object detection models. We also discover a surprising behavior of the CNN-based object detection models: these models sometimes do not use the edges of the object to decide where to draw the bounds of the box even though they accurately locate the object. Finally, we introduce two metrics that quantify how a detector uses the available information in the input image. One metric is called the misapprehension quotient (MAQ) and the other metric is called the tendency to incorporate off-target information (TOffI).

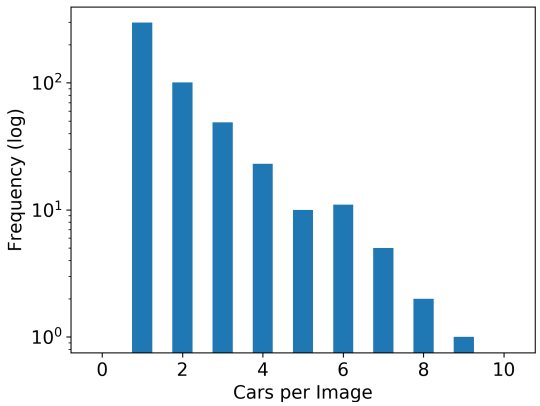
2 Data

The purpose of this section is to provide a general overview of the data. First, we specify from what sources the different parts of the data have been retrieved. Thereafter, some descriptive statistics are provided.

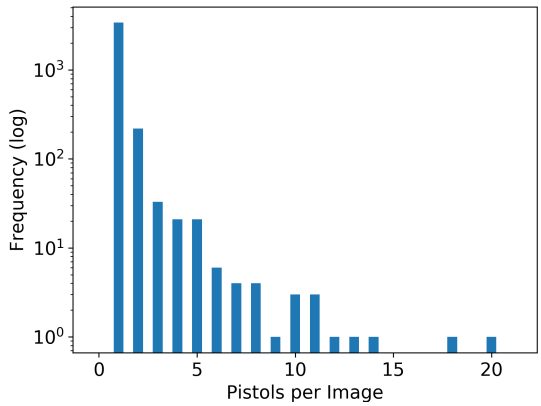
2.1 Data Sources

Labelled images are obtained from the project website associated with the paper of Olmos et al. (2018) [1] and from Google's Open Images Dataset V4 [2]. The former source provides 3,000 annotated images of pistols for training and 608 test images, which are used to compare performance across papers. The latter source provides 651 images of handguns with bounding box annotations. Part of these images (225 images) is used as the extra training data, part of it (75 images) serves as validation data, and the last part (350 images) constitutes the test data. Additionally, 70 unlabeled images containing pistols with a Creative Commons licence are retrieved from Google image search and Flickr. These images have been annotated by hand and are used as additional validation data. In addition to images that contain instances of pistols, 500 annotated traning images from the Open Images Dataset have been included that contain labelled instances of cars. These car images have been added because it turns out that training with pistol images alone yields undesirable inference outputs. A detailed motivation for adding images of cars to the training data is provided in the results section. Finally, 79 unlabeled images of cars with a Creative Commons licence are retrieved from Google image search. These 79 car images are used to check if the

[1] https://sci2s.ugr.es/weapons-detection
[2] https://storage.googleapis.com/openimages/web/index.html



(a) A histogram of the frequency that a certain number of cars appears in a single image.



(b) A histogram of the frequency that a certain number of pistols appears in a single image.

Figure 1: Histograms indicating the frequency of multiple objects of interest appearing in a single image.

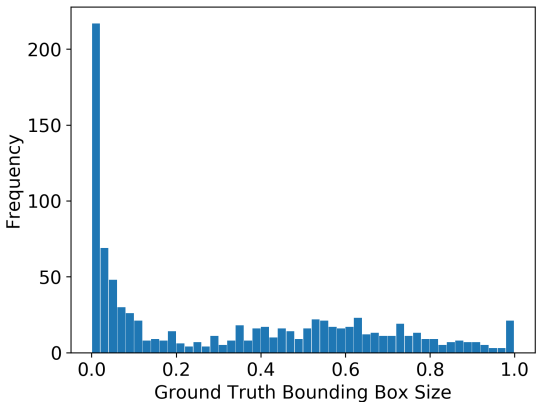
undesirable inference outputs still occur after adding car images to the training data. For the occlusion experiments, we have used images from the test set.

2.2 Descriptive Statistics

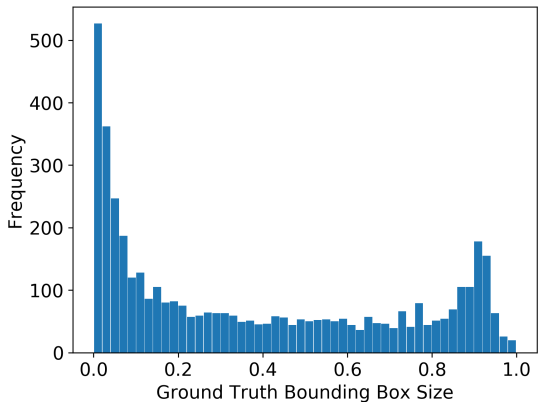
All the labelled images from different sources constitute a dataset that contains 4221 images in total. 500 of these images contain labelled instances of cars and do not feature any pistols. The other 3721 images contain labelled instances of pistols and 24 of these images feature one car or multiple cars in the background. These background cars are not labelled although it would be better if they were. The training set contains a total of 3225 images of pistols and 500 images of cars. The validation set contains a total of 145 images of pistols. The test set contains a total of 350 images of pistols.

On some images multiple pistols or multiple cars are featured. The frequency of these occurrences is summarized in the histograms of Figure 1. To make the histograms more legible a logarithmic scale is used. From Figure 1a and Figure 1b it can be observed that most images contain only a single object. About 300 images contain only one car, about 100 images contain two cars, only two images contain eight cars, and only one image contains nine cars. More than 2000 images contain a single pistol, about 200 images contain two pistols, and exactly a hand full of images contains 12 or more (up until 20) pistols.

Objects that cover only a small part of the image surface are harder to detect than objects that span the whole image. The histograms of Figure 2 provide an impression of the distribution of the object sizes. The size of an object is expressed in terms of the percentage of the image surface covered by its ground truth bounding box. A ground truth bounding box is a rectangular box that tightly encloses an object of interest. Ground truth bounding boxes are drawn, labelled,



(a) Histogram of the frequency with which cars are displayed at different scales.



(b) Histogram of the frequency with which pistols are desplayed at different scales.

Figure 2: Histograms summarizing the distribution of object sizes. The size of an object is measured in terms of the fraction of the image surface that is covered by the ground truth bounding box corresponding to that object.

and validated by humans. From Figure 2a it can be seen that for more than 200 instances of cars the ground truth bounding box covers less than two percent of the image surface. This histogram also shows that within the dataset, cars are displayed at all different scales at which they can be displayed. From Figure 2b it can be observed that for the pistol class, it also holds that ground truth bounding boxes covering less than two percent of the image surface are most common. Though compared to the cars class, the pistol class has relatively fewer examples that are displayed at the smallest scale and has relatively more examples displayed at a medium scale. Moreover, the histogram has a small peak that indicates that there is a relatively large number of objects which cover between 88 and 94 percent of the image surface.

3 Methodology

We devote this section to discussing the details about the detection methods and the setup for the occlusion experiments. Additionally, relevant theoretical topics are explained briefly in this section. These topics include: convolutional neural networks, the Faster R-CNN architecture, the SSD architecture, and the concept of using a cyclical learning rate schedule. The data and our codes needed to reproduce our experiments are available as supplementary material in a GitHub repository.[3]

[3] https://github.com/MartijnCa/occlusion_experiments

3.1 Brief Introduction to Artificial Neural Networks

A neural network is a function that takes a number of variables as input and outputs a set of predictions. With a neural network, arbitrarily complex non-linear relationships between the predictor and the response variables can be modeled. A neural network has to be trained in order to recognize the relationships and yield good predictions. Training is done through a step-wise procedure called gradient descent.

Basic neural networks consist of fully connected layers. Associated with the i^{th} fully connected layer is the input activation vector $a_i = [a_{i,1}, a_{i,2}, \ldots, a_{i,n_i}]^T$, the output activation vector a_{i+1}, the vector of biases $b_i = \left[b_{i,1}, b_{i,2}, \ldots, b_{i,n_{i+1}}\right]^T$, and the weight matrix

$$\mathbf{W}_i = \begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,n_i} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,n_i} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_{i+1},1} & w_{n_{i+1},2} & \ldots & w_{n_{i+1},n_i} \end{bmatrix}, \tag{1}$$

where n_i is the size of activation vector i. A single fully connected layer with $n_i = 3$ input activations and $n_{i+1} = 2$ output activations is presented in Figure 3. This network diagram shows the input activations on the left and the output activations on the right. From each of the input activations, n_{i+1} arrows are pointed towards the elements of the intermediate result vector T_i. To obtain T_i the input activations are multiplied with the appropriate weights and a bias term is added. The elements of T_i are individually passed through the ReLU activation function to yield the output activations, as indicated by the final arrows in the diagram. Since its successful application by Glorot et al. (2011), the ReLU, which stands for rectified linear unit, is a popular choice for the activation function. The ReLU non-linearity, which we also use for the object detectors in this research, is defined as

$$\text{ReLU}(x) = \max(0, x). \tag{2}$$

In general, the first part of the forward pass through a fully connected layer can be compactly denoted in terms of a matrix vector multiplication $T_i = \mathbf{W}_i a_i + b_i$. The resulting intermediary result is passed through an activation function $g()$ to obtain the output activation vector $a_{i+1} = g(T_i)$. Next to the ReLU activation function there are other possibilities, such as the sigmoid function or the identity function.

Multiple layers of different sizes can be stacked on top of each other to form an artificial neural network. In the final layer of the network, the activation function is replaced by an output function $f_k(T_{L,k})$, where $T_{L,k}$ is the k^{th} element of intermediate result vector corresponding to the L^{th} and final layer of a neural network. This function is the final transformation and outputs the predictions. In the context of regression, it is common to use the identity function as the output function. In the context of classification with K categories, it is common to use the softmax

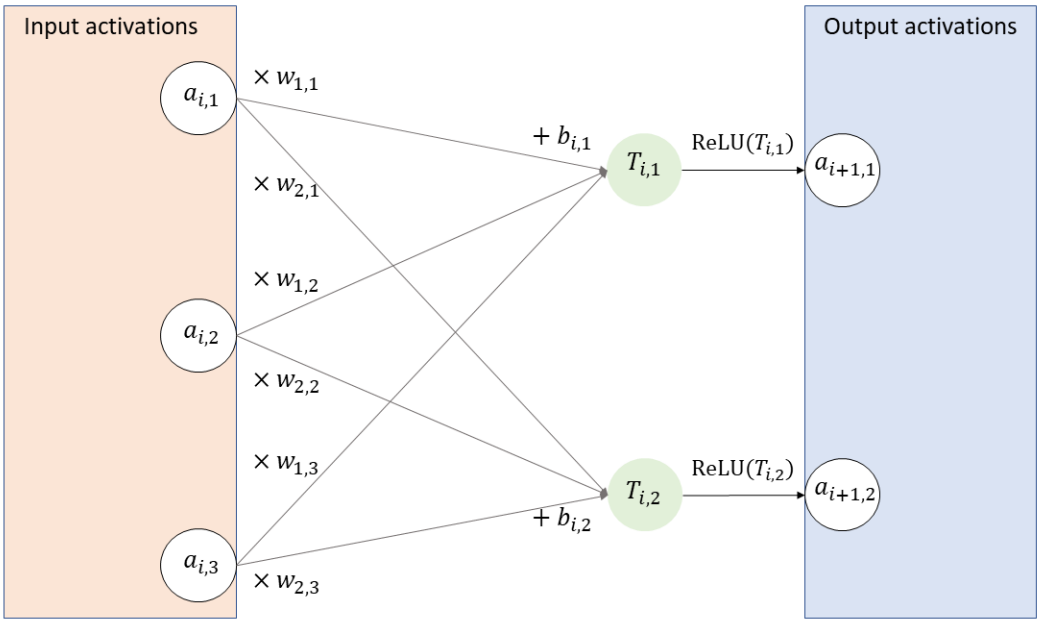


Figure 3: Visualization of a fully connected layer with input activations a_i and output activations a_{i+1}, where $n_i = 3$ and $n_{i+1} = 2$.

function defined as

$$f(T_{L,k}) = \frac{e^{T_{L,k}}}{\sum_{h=1}^{K} e^{T_{L,h}}}. \tag{3}$$

People who are familiar with logistic regression will recognize this function. In fact, some neural networks that have the softmax function as its output function are equivalent to a logistic regression model.

Training

The goal of training a neural network is to find the model parameters, weights and biases, that correspond to a maximal or close to maximal model fit. The model fit is quantified through the loss function. The further away a prediction is from the target, the higher the loss contribution of this training example will be. A commonly used loss function in the context of regression is the sum of squared errors, which is called the L_2 loss. An alternative is the sum of absolute errors or L_1 loss. In the context of classification, the cross-entropy loss is used most frequently, which is defined as

$$L_{\text{cross-entropy}} = \sum_{j=1}^{N} \sum_{k=1}^{K} p_{j,k}^{t} \log\left(p_{j,k}^{p}\right), \tag{4}$$

where $p_{j,k}^{p}$ corresponds to the predicted probability of example j belonging to class k, $p_{j,k}^{t}$ is a binary indicator of true class membership, N is the total number of examples in the training set,

and K is the total number of classes.

The forward pass through a neural network yields the predictions and a loss function evaluation. A backward pass through a neural network entails using the chain-rule to differentiate with respect to the model parameters. Performing a backward pass through a network is known as backpropagation. The backward pass yields the gradient and the gradient corresponds to the direction of steepest increase. The idea behind gradient descent is to move in the direction of the steepest decrease. So, the parameters are updated by subtracting a factor, the learning rate, times the gradient from the parameter vector. The learning rate is a hyper-parameter for which the value can be decided upon through tuning.

3.2 Convolutional Neural Networks

The fully connected layer is the standard and original building block for neural networks. As its name suggests, each of the neurons in a fully connected layer is connected to all the neurons of the previous layer. For image classification, a network that consists only of fully connected layers is not ideal. Such networks can classify handwritten digits reasonably well. However, such networks are generally not able to recognize more complex visual structures such as the appearance of a dog or cat. A different kind of neural network, the convolutional neural network, has proven to be capable of performing increasingly complex visual recognition tasks successfully.

As opposed to a network with fully connected layers, a convolutional neural network is designed to take the spatial structure of an image into account. Convolutional layers are characterized by their local connectivity. Each neuron that belongs to the first convolutional layer is caused to activate (or not) by only a specific group of adjacent pixels in the input image. Each neuron that belongs to the second convolutional layer is caused to activate (or not) by only a specific group of adjacent activations in the first convolutional layer and so forth. Convolutional neural networks are designed to first look at all different small parts of the image separately and only later combine the information contained in different parts of the image to make predictions about the image content.

Convolutional layers are most vividly understood in visual terms. Therefore, we explain how convolutional layers work by walking through a pair of convolutional layers, which are schematically represented in Figure 4. The leftmost rectangular grey box corresponds to the array representation of an RGB input image containing 227×227 pixels. For each of the pixels there are three different values specifying the intensity of the colors red, green, and blue. So, the array has height $H_0 = 227$, width $W_0 = 227$, and a depth $D_0 = 3$, where the last dimension is referred to as the number of channels. This 3-dimensional array is the input activation map to the first convolutional layer, which outputs an activation map of size $H_1 \times W_1 \times D_1$, where H_1, W_1, and D_1 refer to the height, width, and depth of the activation map, respectively. In the example of Figure 4, the output activation map that belongs to the first convolutional layer corresponds to the middle grey

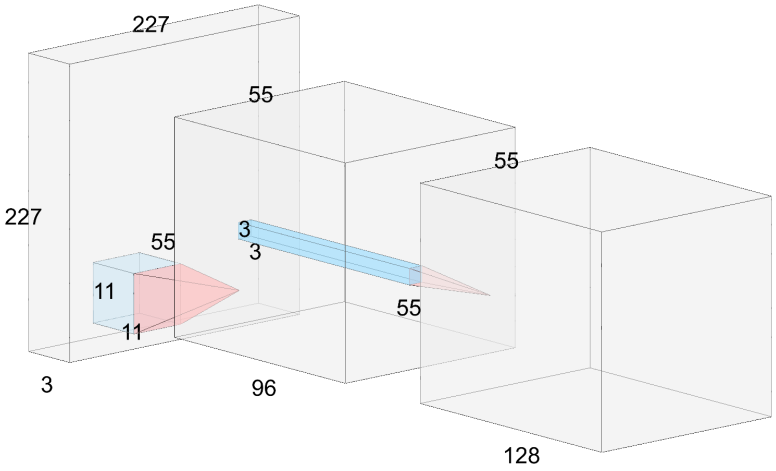


Figure 4: A visual representation of a set of convolutional layers. Three activation maps and two layers are displayed. The stride corresponding to the first layer is $s_{1,v} = s_{1,h} = 4$ and the stride corresponding to the second layer is $s_{2,v} = s_{2,h} = 1$. For the second layer, zero padding of one is applied.

rectangular box. The associated dimensions are $H_1 = W_1 = 55$ and $D_1 = 96$.

Like all convolutional layers, the first convolutional layer consists of a set of convolutional filters. Each convolutional filter consists of a 3-dimensional array of weights and a bias term. The weight array has dimensions $A_{1,h} \times A_{1,w} \times D_0 = 11 \times 11 \times 3$, where $A_{1,h} \times A_{1,w}$ is the spatial extent of the filter and D_0 the depth. The depth of a convolutional filter has to be equal to the number of channels of the input activation map. So, each filter in the first convolutional layer of any network that takes RGB images as input must have a depth of three.

Convolutional filters are multiplied with patches from the input activation map to obtain output activations. The multiplication of one convolutional filter which belongs to the first convolutional layer, with a specific patch from the input activation map is visualized in the diagram as a blue rectangular box with a red pyramid like shape on top. The red pyramid illustrates that the multiplication produces a single scalar output. After adding the bias term to this scalar, an output activation is obtained.

In general, the multiplications associated with the i^{th} layer are carried out as follows. Let $\mathbf{a}_{\text{in}}$ refer to any given patch of the input activation map, which is a $A_{i,h} \times A_{i,w} \times D_{i-1}$ array of activations, let W_i refer to the weight array of one of the convolutional filters, which has dimensions $A_{i,h} \times A_{i,w} \times D_{i-1}$, and let b_i refer to the scalar bias term belonging to the filter. The following dot product yields the above mentioned scalar output activation

$$a_{\text{out}} = \text{vec}(\mathbf{a}_{\text{in}}) \cdot \text{vec}(\mathbf{W}_i) + b_i, \tag{5}$$

where the vec() function vectorizes an array argument. Typically, the output activations are passed through an element-wise activation function such as the ReLU function.

All filters slide over the input activation map to obtain output activations at all locations. The filters are first multiplied with the $A_{i,h} \times A_{i,w}$ patch in the top left corner of the input activation map. Subsequently, the filters are multiplied with a patch, which is located $s_{i,h}$ cells to the right of the previous patch, where $s_{i,h}$ and $s_{i,v}$ refer to the horizontal and vertical stride respectively. The filters keep sliding to the right until the right edge of the input activation map is reached. Next, the filters are slid $s_{i,v}$ cells down and start sliding from left to right again. At each location a single filter outputs a scalar value. When a filter has slid all the way to the bottom right corner of the input activation map, it has output a 2-dimensional grid of activations, which constitutes one of the channels of the output activation map. Every filter is responsible for one output channel.

The spatial extent and stride of the convolutional filters have to be compatible with the dimensions of the input activation map. A situation in which a filter partially slides out of an image is not permitted. It is possible to add zeros to all the edges of an activation map, which is called zero padding. A common purpose of zero padding is ensuring that the subsequent activation map has the same dimensions as the previous one. From the above explained procedure it can be deduced that for an input activation map with dimensions $H_{i-1} \times W_{i-1} \times D_{i-1}$ the output activation map has dimensions

$$\begin{aligned} H_i &= \left(H_{i-1} - A_{i,h} + 2P_{i-1}\right)/s_{i,v} + 1, \\ W_i &= \left(W_{i-1} - A_{i,w} + 2P_{i-1}\right)/s_{i,h} + 1, \quad \text{and} \\ D_i &= F_i, \end{aligned} \tag{6}$$

where F_i indicates the number of convolutional filters of layer i and P_{i-1} the number of zeros that are padded to the input activation map.

The formula's of Equation (6) can be used to reason about the dimensions of the activation maps and convolutional filters displayed in Figure 4. The height and width of the second activation map follow from computing $(227 - 11)/4 + 1 = 55$ and those of the third activation map from computing $(55 - 3 + 2 \cdot 1)/1 + 1 = 55$. From the fact that the second activation map is 96 channels deep (see Figure 4) and the fact that the multiplications associated with one sliding filter result in a single channel of output activations (the third equality of Equation (6)), it follows that the first convolutional layer must consist of 96 filters. Similarly, the second convolutional layer must consist of 126 filters. As can be observed from the diagram, every weight array that belongs to a filter of the second convolutional layer has dimensions $A_{2,h} \times A_{2,w} \times D_1 = 3 \times 3 \times 96$.

The explanation above, which involves sliding filters, is a conceptual explanation. In practice, all the computation involved with a convolutional layer is done through a single matrix multiplication. Finally, it is worth noting that convolutional layers need far fewer parameters than fully connected layers to output the same number of activations.

3.3 Object Detection

This subsection is devoted to explaining the Faster R-CNN and SSD models. The explanation includes an overview of the model architectures, the loss function specifications, and the performance evaluation measures.

Faster R-CNN

A Faster R-CNN model consists of two neural networks: a region proposal network (RPN), and a joint object classification and bounding box regression network, which is the same as a Fast R-CNN (Girshick, 2015). The first set of layers for both networks is a shared set of convolutional layers. After an image is passed through the convolutional layers the result is a feature map. Therefore, such a set of convolutional layers is also referred to as the feature extractor. For the feature extractor it is common to use a number of the convolutional layers of an image classification model such as VGG-16 (Simonyan & Zisserman, 2014), ResNet (He et al., 2016), or Inception-v2 (Szegedy et al., 2015). The layers after the feature extractor are different for the RPN and the Fast R-CNN. An explanation of the two different networks that comprise a Faster R-CNN model is provided in the following paragraphs.

The purpose of the RPN is to produce region proposals and to do so, anchor boxes are used. Anchor boxes are rectangular regions with different aspect ratios and scales that are regularly tiled over the feature map. Typically, three different scales and three different aspect ratios are used, resulting in nine different types of anchor boxes per location. Figure 5a illustrates that there are k different anchor boxes belonging to every location in the feature map. The above mentioned regular tiling entails that the red sliding window in the same diagram reaches all possible 3×3 grids in the convolutional feature map.

The RPN consists solely of convolutional layers and is composed of the layers of the feature extractor and two convolutional layers on top of the feature map. The final convolutional layers, the layers on top of the feature map, can be more easily understood by considering these convolutional layers to be a collection of smaller networks, an individual network for each anchor box. Figure 5a visualizes such a mini-network. Each anchor box can potentially be modified into a region proposal by a mini-network. For each anchor box at a specific location, the mini-network simultaneously outputs an objectness score and bounding box offsets. The bounding box offsets provide adjustments to the location and size of the anchor box and the objectness score represents the model's prediction of the likelihood that there is an object present with which the box has high overlap. Typically, the 300 boxes with the highest objectness score are used as the region proposals and the others are discarded.

The parameters of the mini-networks are different across anchors of different scales and aspect ratios but are shared across image locations. The first layer consists of a 3×3 convolution with 256 filters, takes as input a 3×3 grid in the feature map, and outputs a 256-dimensional vector,

which is presented as the intermediate layer in Figure 5a. The 256-dimensional vector is the input for a fully connected layer with six neurons. These six neurons make up the last layer of the network, which can be split into two parts with distinct purposes. One part, consisting of four neurons, represents a set of predictions for the bounding box offset denoted as $t^p = \{t^p_x, t^p_y, t^p_w, t^p_h\}$. The other two neurons are the input of a softmax activation function that outputs the objectness score, p^p. The multitude of mini-networks can be rewritten as and amounts to a single network of two convolutional layers on top of the feature map.

For training a RPN it is necessary to label anchors by trying to match them with ground truth bounding boxes. All objects of interest in the training data are provided with a ground truth box. For each anchor it is judged whether or not it sufficiently overlaps with one of these ground truth boxes so that the anchor can be matched to the ground truth box. A selection of anchors is matched and to this selection a positive binary label, $p^t = 1$, is assigned, which indicates that the anchor is considered to contain an object of interest. To a different selection of anchors a negative binary label, $p^t = 0$, is assigned indicating that this anchor contains no object or a very small part of an object. There are also a number of anchors that receive neither a positive nor a negative label.

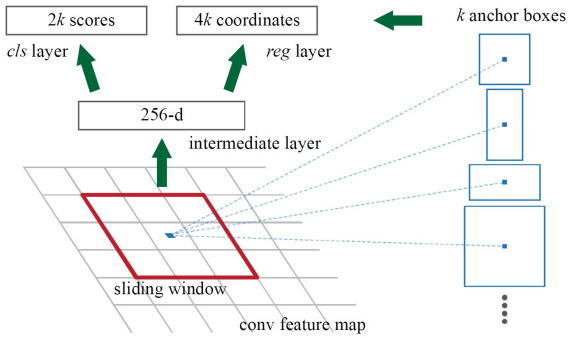
An anchor is considered to contain an object and is assigned a positive label, $p^t = 1$, if the overlap between the anchor box and any ground truth box, measured in terms of intersection-over-union (IoU), is 0.7 or higher. The intersection-over-union measure, also known as Jaccard overlap, will be defined at a later stage of the methodology section. It is possible that for a specific ground truth bounding box, b, there exist no anchor box that has an IoU of 0.7 or higher with b. In a rare case like that, a positive label is assigned to the anchor box that has the highest IoU with b and this anchor box is matched with the ground truth box. The other positive anchors, which are not matched in this way, are matched with the ground truth box that they overlap with the most. An anchor is assigned a negative label if (i) it is not assigned a positive label *and* (ii) the anchor has an IoU lower than 0.3 with all ground truth bounding boxes. Anchors that are assigned neither a positive nor a negative label do not contribute to the loss function evaluation.

If an anchor is assigned a positive label then it is also assigned a set of ground truth box offset scores $t^t = \{t^t_x, t^t_y, t^t_w, t^t_h\}$. The true and predicted offset scores are defined as

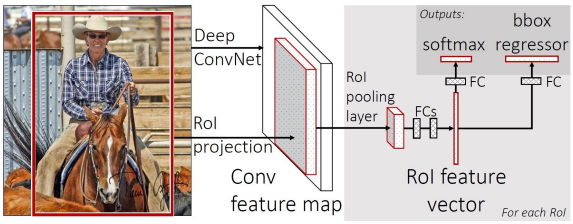
$$\begin{aligned} t^t_x &= (x^t - x_a)/w_a, \quad & t^t_y &= (y^t - y_a)/h_a, \\ t^t_w &= \log(w^t/w_a), & t^t_h &= \log(h^t/h_a), \\ t^p_x &= (x^p - x_a)/w_a, & t^p_y &= (y^p - y_a)/h_a, \\ t^p_w &= \log(w^p/w_a), & t^p_h &= \log(h^p/h_a), \end{aligned} \tag{7}$$

where x^p, y^p, h^p, and w^p are the coordinates of the upper left corner of the predicted box and the height and width respectively. The subscript a stands for "anchor" and the superscript t stand for "true".

The contribution to the loss of a single image is obtained by summing over the loss contribution,



(a) Visualization of a single mini-network. The RPN layers on top of the feature map are a collection of these mini-networks. On the right it is illustrated that k different anchor boxes are considered for each 3×3 grid on the feature map.



(b) Visualization of the layers that a single region proposal is passed through to produce bounding box offsets and softmax probabilities per class as output.

Figure 5: Visual representations of the two different components the Faster R-CNN architecture consists of. The image on the right has been retrieved from Girshick (2015) and the image on the left has been retrieved from Ren et al. (2015).

L_i, of every anchor i. The contribution to the loss of a single positively or negatively labelled anchor is a multitask loss function

$$L_i(p^p, p^t, t^p, t^t) = L_{\text{cls}}(p^p, p^t) + \lambda p^t L_{\text{loc}}(t^p, t^t), \tag{8}$$

where we choose $\lambda = 1$. The localization loss is defined as

$$L_{\text{loc}}(t^p, t^t) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i^p - t_i^t), \tag{9}$$

where

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2, & |x| < 1, \\ |x| - 0.5, & |x| \geq 1. \end{cases} \tag{10}$$

The log loss function, which is the same as cross-entropy over two classes, is chosen as the classification loss

$$L_{cls}(p^p, p^t) = -p^t \log(p^p) - (1 - p^t) \log(1 - p^p). \tag{11}$$

Apart from removing normalization terms from the multitask loss of Equation (8), the loss function specifications presented here are identical to what is used by Ren et al. (2015). The normalization terms are removed because Ren et al. (2015) empirically show that including these terms is unnecessary.

Next to the RPN module, a Faster R-CNN model consists of a second module. This second module is a neural network that produces classification scores for the region proposals and class specific bounding box offsets relative to the region proposal boxes. The second module is the same as the detection network of Fast R-CNN and the layers that make up this module are visualized by the diagram of Figure 5b. As is the case for the RPN module, a feature extractor comprises the first layers of the Fast R-CNN module. In the diagram, the "Deep ConvNet" arrow refers to the feature extractor that outputs a convolutional feature map.

For a given region proposal, its projection on the feature map is cropped from the feature map and this crop of features is passed through the region of interest (RoI) pooling layer. The purpose of this layer is to resize a crop from the feature map of any size or aspect ratio into an activation map with fixed dimensions $W \times H$. The RoI pooling layer is similar to a conventional pooling layer, except for the alternative way in which the input activation map is divided into a grid. Namely for the RoI pooling layer, the crop of the feature map with dimensions $w \times h$ is divided into a grid with cells that approximately have size $w/W \times h/H$. In each of the grid cells the maximum value is recorded and passed on to the next layer, which ends up having dimensions $W \times H$. Similarly to conventional pooling, each channel is pooled separately.

After the RoI pooling layer, a sequence of fully connected layers follows, which is displayed in the diagram of Figure 5b right before the RoI feature vector. After the RoI feature vector comes the final layer of the Fast R-CNN module. Similarly to the RPN, the final layer of the Fast R-CNN consists of two parts, one part for regression and one part for classification. The classification part produces softmax scores for each class including a background class. The regression part produces class specific bounding box offsets.

The loss function associated with Fast R-CNN is similar to the multitask loss of Equation (8). The classification loss is a cross-entropy loss over softmax probabilities for all classes and a background class, which amounts to the "pistol", "car", and "background" classes for the current paper. If the ground truth class is not background, then we add the localization loss of Equation (9), where the predicted box offset scores are now specific to the ground truth class. In the current context, the λ in Equation (8) is also chosen to be equal to 1. The RPN and the Fast R-CNN networks are trained simultaneously through a procedure called approximate joint training (Ren et al., 2015).

During detection, the image is passed through the model and for each of the 300 region proposals generated by the RPN, the model outputs a per class classification confidence and class specific refinements to the proposed bounding boxes. Subsequently, non-maximum suppression is applied for each object class independently, to avoid duplicate detections. Non-maximum suppression entails that predictions are discarded which have a high overlap with a different more confident prediction of the same class. We use IoU > 0.6 as the suppression threshold.

SSD

As opposed to Faster R-CNN, the SSD architecture involves no region proposal step and contains no RoI pooling layer. The SSD model is a unified network that outputs classification confidence scores and bounding box offsets for default boxes. Default boxes in the context of SSD are the equivalent of anchor boxes in the context of Faster R-CNN. Instead of using anchor boxes to generate region proposals and refining the region proposals to obtain final bounding box predictions, the SSD model predicts bounding box offsets directly from the default boxes without an intermediate step. Performing computations for all default boxes simultaneously instead of sequentially passing region proposals through a set of fully connected layers, allows for faster detection. Additionally, skipping the region proposal generation step saves computation time. However, with the faster inference time of SSD, generally comes a lower detection accuracy.

The architecture of the SSD model is visually represented in Figure 6. From this image it can be seen that a feature extractor makes up the first layers of the model, as is the case for the Faster R-CNN model. Liu et al. (2016) use the first convolutional layers of the VGG-16 model as feature extractor for the original SSD model, which is shown in the diagram, whereas for this research the Inception-v2 model is used for feature extraction. On top of the feature map there is a sequence of convolutional layers. The dimensions of the filters that belong to these layers are provided on the bottom of the diagram of Figure 6, where "Conv: $3 \times 3 \times 512 - \text{s2}$" means that the spatial extent of a filter is 3×3, the number of filters is 512, and the stride is two. Reasoning about the dimensions of the intermediate activation maps can be done using the formulas of Equation (6). For example, between "Conv7" and "Conv8_2" there are two convolutional layers. The first one has 256 filters of size $1 \times 1 \times 1024$ and a horizontal as well as vertical stride of one, and the second one has 512 filters of size $3 \times 3 \times 256$ and a horizontal as well as vertical stride of two. For the first convolutional layer, no zero padding is applied and for the second convolutional layer, zero padding of one is applied. The dimensions of the intermediate feature map between "Conv7" and "Conv8_2", $H_{\text{int}} \times W_{\text{int}} \times D_{\text{int}}$, can be computed as

$$\begin{aligned} H_{\text{int}} &= (19 - 1 + 2 \cdot 0)/1 + 1 = 19, \\ W_{\text{int}} &= (19 - 1 + 2 \cdot 0)/1 + 1 = 19, \\ D_{\text{int}} &= 256, \end{aligned} \tag{12}$$

and the dimensions of the feature map of "Conv8_2" can be computed as

$$\begin{aligned} H_{\text{Conv8\_2}} &= (19 - 3 + 2 \cdot 1)/2 + 1 = 10, \\ W_{\text{Conv8\_2}} &= (19 - 3 + 2 \cdot 1)/2 + 1 = 10, \\ D_{\text{Conv8\_2}} &= 512. \end{aligned} \tag{13}$$

It is worth noting that it is not mentioned in Figure 6 how much zero padding has been applied for the different layers. However, this can be inferred from the other information that is presented

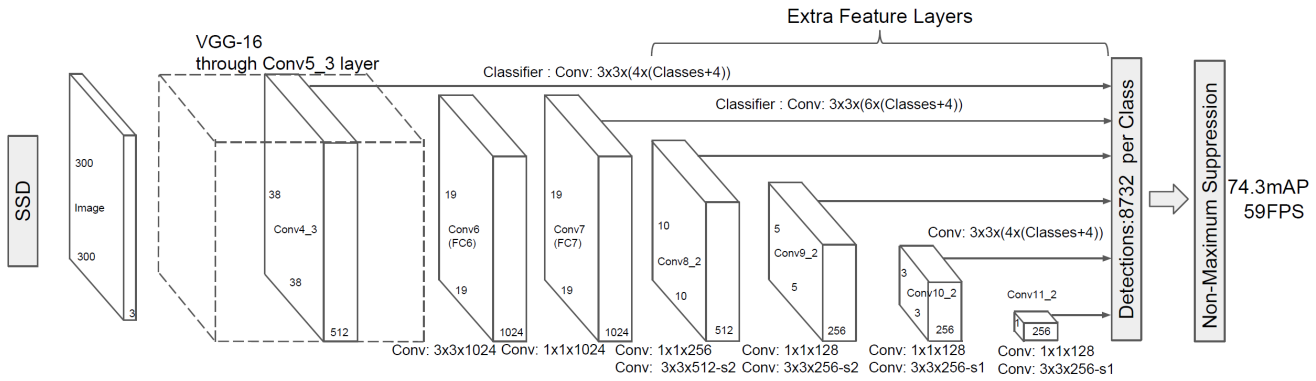


Figure 6: Visualization of the SSD architecture. This image has been retrieved from the work of Liu et al. (2016).

along with the figure.

The final layers successively decrease in size, and layers of different sizes are connected to the output layer, which is represented in Figure 6 by the arrows that point to the detection output layer. By connecting these feature maps of different sizes to the output layer the model is better able to detect objects of different sizes. From the diagram it can be observed that there is also a layer early in the network that is connected to the output layer. Doing so can be beneficial because lower level activation maps can capture more detailed features. Long et al. (2016) have shown that connecting lower level feature maps to the output layer is useful in the context of semantic segmentation, which is part of the motivation why Liu et al. (2016) connect the layer early in the network to the output layer.

Prediction outputs from all different feature layers are generated in convolutional fashion. The elementary unit is a convolutional filter of size $3 \times 3 \times p$ which corresponds to a certain default box aspect ratio and produces either a classification score for an object category or a box offset estimate. This filter is convolved over a feature layer of dimensions $m \times n \times p$, where p is the number of channels of the feature layer. The filter produces a prediction at all $m \times n$ locations. To make it possible to reach all edges of the feature layer grid with convolutional filters of size 3×3, zero padding of one is applied. Choosing to use k different aspect ratios yields k different default boxes at each location. For each default box, 4 box offsets have to be predicted and object category scores have to be output for all c categories (including background). So, having k different default boxes at each feature map location implies that $(c + 4) \cdot k$ filters are needed. A total of $(c + 4) \cdot k \cdot m \cdot n$ outputs are produced for a single feature map of size $m \times n$.

The bounding box offsets predicted by the SSD model are class independent, whereas in the context of Faster R-CNN those offsets are class specific. Per feature map location there are default boxes of different aspect ratios. A default box at a certain location in the feature map is used to detect objects in the same relative location in the image. The receptive field of an individual feature in a feature map higher up in the network is (or almost is) the whole image. So, a feature

in one location of the feature map can be caused to activate by pixels in a completely different location in the actual image. Liu et al. (2016) claim that the activations in a certain region of a feature map learn to be responsive to objects displayed at a certain scale in the same relative region of the image. The specific scale that a feature map is training to be responsive to corresponds to the scale of the default boxes tiled over it.

The scale of the default boxes that are tiled over feature map k is computed as

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1), \quad k \in [1, m], \tag{14}$$

where s_{min} and s_{max} are the lowest and highest scales corresponding to the first and last feature layer, respectively. In Equation (14), the scalar m indicates the total number of feature layers. For the default boxes, a set of five different aspect ratios, $\{1, 2, 5, 1/2, 1/3\}$ is used. The height, h_k^a, and width, w_k^a, of a default box with aspect ratio a_r in feature layer k are computed as

$$\begin{aligned} h_k^a &= s_k\sqrt{a_r}, \\ w_k^a &= s_k/\sqrt{a_r}. \end{aligned} \tag{15}$$

There is one additional default box of aspect ratio one, which has a scale of $\sqrt{s_k s_{k+1}}$. The total number of boxes per location in a feature map adds up to six.

Ground truth boxes need to be matched with default boxes in order to be able to compute the loss associated with a prediction produced for a specific default box. Several rules are applied to obtain matches. The first one being that for each ground truth bounding box, the default box that has the highest overlap in terms of IoU is assigned to match with that ground truth box. Additionally, each default box that has not been matched yet, is matched to a ground truth box if the IoU between the boxes is 0.5 or higher. If a default box has an IoU of 0.5 or higher with multiple different ground truth boxes then it is assigned to match the ground truth box that it is overlapping with the most.

The training objective for the SSD model is similar to the training objective used for Faster R-CNN. The loss is a weighted sum between the localization and classification loss, as in Equation (8). The contribution to the loss function of a single image is computed as a weighted sum over the contributions of all default boxes. The contribution per image is weighted proportional to the number of default boxes that have a match with a ground truth box. The localization error penalty is quantified through a smooth_{L_1} loss with parameterizations of the box offsets as defined in Equation (7). For classification, a multiclass cross entropy loss function is employed over a set of estimated class probabilities resulting from a softmax function.

Typically, a large percentage of the default boxes is not matched to any ground truth bounding box. Liu et al. (2016) find that it is beneficial to apply a methodology called hard negative mining. Namely, this approach leads to faster optimization and more stable training. Hard negative mining entails taking only a part of the unmatched default boxes into account. The negative examples are

ordered descendingly according to the magnitude of the classification loss. The negative examples at the top of the ordering are included in the loss function evaluation and the rest of the negative examples do not contribute to the loss and do not influence the parameter updates. The amount of included negative examples is chosen in such a way that there are no more than three times as many negative examples as positive ones.

At detection time, a forward pass though the network is performed and non-maximum suppression with an IoU threshold of 0.6 is applied to avoid duplicate detections.

Performance Evaluation Measures

For evaluating the localization accuracy, the intersection-over-union (IoU) measure for overlap is used. The IoU between predicted bounding box B_p and ground truth bounding box B_{gt} is computed by the formula

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}. \tag{16}$$

The overall performance of an object detection method is commonly measured in terms of AP (Everingham et al., 2010). The AP metric summarizes the precision-recall curve by averaging over the interpolated precision measured at 11 different recall levels. To obtain the precision recall curve the predictions are ordered descendingly and ranked according to the prediction confidence. Sequentially, one considers the single highest ranked prediction, the two highest ranked predictions, the three highest ranked predictions and so forth. When considering the n highest ranked predictions one computes the recall as $r = TP/P$ and the precision at recall level r as $p(r) = TP/n$, where TP indicates the number of true positives within the set of n highest ranking predictions and P indicates the total number of positive examples in the sample. To draw the precision-recall curve, the combinations of precision and recall for each value of n are plotted in a graph, while connecting the dots. The interpolated precision at recall level r, $p_{interp}(r)$, is equal to the highest precision that corresponds to a recall value larger or equal to r, which amounts to

$$p_{interp}(r) = \max_{\tilde{r}, \tilde{r} > r} p(\tilde{r}). \tag{17}$$

The formula for computing the AP amounts to

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, 0.2, \ldots, 1\}} p_{interp}(r). \tag{18}$$

Traditionally in the context of computing the AP, a detection is regarded as a true positive if the IoU between the predicted box and the ground truth box is larger than 0.5 and the predicted label corresponds to the ground truth label. The AP measure reported here is different and corresponds to the primary challenge metric of the COCO competition.[4] This variation of the

[4] http://cocodataset.org/#detection-eval

AP measure is calculated by averaging over different AP_{IoU} values computed using different IoU thresholds for true positive determination. More specifically, a set of ten different AP_{IoU} values $\{AP_{0.5}, AP_{0.55}, AP_{0.60}, \ldots, AP_{0.95}\}$ is computed and the average over the elements of the set is computed to obtain the final and reported AP metric.

3.4 Cross-Validation for Estimating Generalization Performance

The models that are trained for this thesis are not only subject to an interpretative analysis, but we also estimate their generalization performance with high accuracy. Computing the AP using only a single holdout sample has drawbacks. Depending on the size of the holdout sample, such an estimate is possibly unreliable, because it can be influenced by contingencies. For example, a test split might happen to contain a disproportionate amount of object instances that are easier or harder to detect. In other words, the variance of an estimate computed using a single holdout sample can be too high. Therefore, we choose to estimate the generalization performance using k-fold cross-validation.

When evaluating the performance using a single holdout sample, choosing the size of the training set and the test set involves a trade-off between the pessimistic bias and the variance of the generalization performance estimate. Using more data for the computation of the generalization performance results in an estimate with a lower variance. However, the generalization performance has to be estimated using data that are new to the model, data that it has not seen before. So, all data that are used for evaluation cannot be used for training. Using less data for training generally yields an inferior model, a model that is pessimistically biased with respect to a model that has been trained on the full set of available data. Exceptions to this general tendency include models that have reached their capacity. [5]

The stringency of the bias-variance trade-off can be alleviated by the use of k-fold cross-validation. At the cost of extra computational overhead, both the pessimistic bias and the variance of the generalization performance estimate can be kept low. A detailed and comprehensive discussion on the bias-variance trade-off and the cross-validation methodology is provided by Raschka (2018).

To perform k-fold cross-validation, the data are randomly divided into k subsets or folds of equal size. Throughout the process, k models are trained and evaluated. Each of the models is trained on $k-1$ folds and evaluated on the remaining fold. The evaluation metrics are averaged over all the evaluation runs to obtain the final generalization performance estimate. This way, each of the available observations is used once for model evaluation, keeping the variance of the performance estimate low.

For larger k the training splits are bigger. Hence, the pessimistic bias is smaller. The ex-

[5] A model that has reached its capacity is a model that has had enough training data at its disposal. Such a model does not benefit from more training data.

treme case is called leave one out cross-validation (LOOCV), which entails that each observation constitutes a fold. LOOCV yields the minimal pessimistic bias but is the most computationally expensive. Another downside is a higher variance of the estimator. A theoretical explanation for this higher variance is provided by Hastie et al. (2009), who state:

> With $K = N$, the cross-validation estimator is approximately unbiased for the true (expected) prediction error, but can have high variance because the N "training sets" are so similar to one another.

Considering this drawback and the computational burden associated with a high value of k we opt for a smaller value of $k = 5$, which is a commonly used value for k. However, small values of k like $k = 5$ (but more so $k = 2$ or $k = 3$) can also lead to an increased variance due to random sampling effects. For this reason, a k of 10 is likely to yield a better estimate of the generalization performance. However, to keep computational burden within reasonable limits we prefer $k = 5$ over $k = 10$, since training a pistol detector takes several hours on a GPU accelerated machine.

In our application the cross-validation is organized as follows. The collections of data that we have previously referred to as the training, validation, and test set constitute the complete set of available data. This complete set contains images featuring cars and images featuring handguns. Throughout the cross validation, the labelled images featuring cars are always used for training. The purpose of including these images of cars in the training sample is to enable the model to learn the difference between cars and handguns. In the results section, more details are provided concerning the necessity to add images featuring cars to the training data. The validation and test set do not contain images that feature labelled cars. So during tuning, it is not taken into account whether a model tends to spuriously detect pistols where there are cars. At test time, this possible shortcoming of a model is also not taken into account when the AP is calculated. If (some of) the models tend to yield such spurious handgun detections, then the estimated generalization performance would be a lot less meaningful. Therefore, we expose all ten models trained for the cross-validation, the five SSD models and the five Faster R-CNN models, to an extra set of 79 unannotated images that feature one or more cars and test for the occurrence of spurious detections.

The images that feature handguns are split into five folds. Each of the five folds is used for evaluation once and the other four folds are used for training together with the car images. Every model is trained using a cyclical learning rate schedule, which is a concept that will be discussed in the next subsection. Before training, the learning rate is tuned. If fold 1 is used for evaluation, then fold 5 is used as validation sample for tuning. The same holds for folds 2 and 1, 3 and 2, 4 and 3, and 5 and 4. How the data are split up into folds is schematically visualized in Figure 7. For the SSD specification as well as the Faster R-CCN specification, the learning rate tuning is repeated five times throughout the cross-validation to ensure no information is leaking from the evaluation samples into the models.

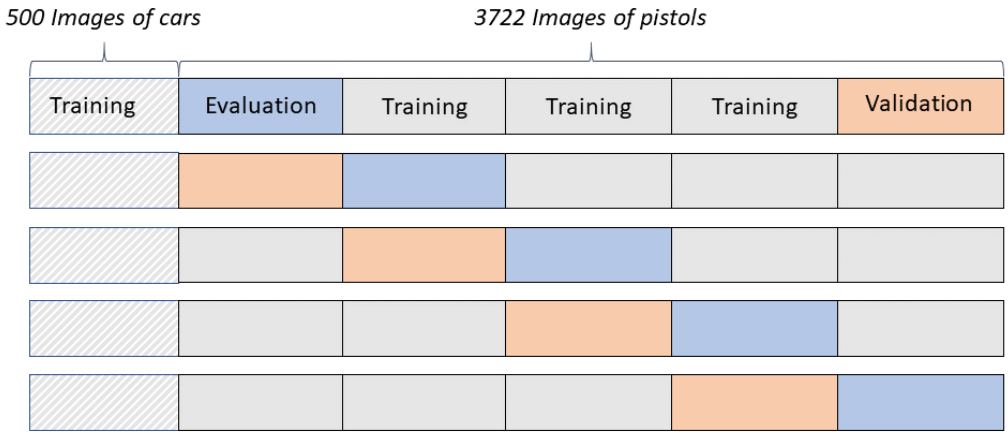


Figure 7: Visual representation of the division of the complete set of available data into folds.

3.5 Model Implementation

Training deep convolutional neural networks is very computationally intensive. Therefore, such models are generally trained using a graphics processing unit (GPU) or a tensor processing unit (TPU). GPUs and TPUs allow for much faster model training and model inference. For our research we used a NVIDIA Tesla K80 GPU. Using this GPU the models train 15 times as fast as they would when using a Intel Xeon CPU with a clock rate of 2.2 GHz. For the model implementation we use the programming language Python and the machine learning and deep learning framework TensorFlow. TensorFlow relies on the NVIDIA CUDA Deep Neural Network library (cuDNN), which is implemented on top of the CUDA software layer. CUDA is an application programming interface (API) that gives access to the parallel computing capabilities of the GPU. The minimum GPU requirements for a NVIDIA GPU are that it must have Compute Capability 3.0 or above and that it needs to have sufficient dedicated GPU memory. To reproduce the results of this paper about 4 GB of dedicated memory should be enough. However, more memory is beneficial because it allows for training with a larger batch size.

Huang et al. (2017) compare different object detection model architectures in terms of speed, accuracy, and required GPU memory. In their paper, three different meta-architectures are combined with a variety of feature extractors. Among these three meta-architectures, Faster R-CNN and SSD are the two that are used in our research. Huang et al. (2017) report as findings that the SSD meta-architecture combined with the Inception-v2 feature extractor yields a relatively high accuracy for a fast model. This makes it a good candidate for the interpretative analysis conducted here. In addition to SSD with Inception-v2, we also use Inception-v2 as feature extractor for the Faster R-CNN model. It is convenient that with quick inference conducting occlusion experiments becomes less time-consuming. Other than this, the choice to use relatively fast models is somewhat arbitrary. Although combining Faster R-CNN with Inception-v2 does not yield the best trade-off

between accuracy and speed (Huang et al., 2017), we choose to use the same feature extractor for both meta-architectures. This way, the possible differences between the meta-architectures, rather than the differences between the feature extractors, can be emphasized when interpreting the inferences. For the sake of briefness and clarity, we do not mention the feature extractor and just name a model by its respective meta-architecture in the remaining part of this paper.

For the implementation, we have used the open-source framework that was built in TensorFlow by Huang et al. (2017).[6] The parameters of both models are initialized with the parameters from the models trained on the COCO dataset. These pre-trained models are retrieved from the same open-source project mentioned above. After initialization, the models are trained to detect pistols and cars. It turns out that it is necessary to add the car class explicitly. Models that we have trained to detect pistols as the only object category tend to mistake cars for handguns, which will be discussed in the results section.

The images are resized before they are passed through the networks for detection. For the Faster R-CNN model an image is resized in such a way that the aspect ratio does *not* change. First, the shortest side is set equal to 600 pixels and if after that the longest side exceeds 1024 pixels, then the longest side is set equal to 1024 pixels, which implies that the length of shortest side will decrease accordingly (to a length shorter than 600 pixels). The three different scales of anchor boxes used for Faster R-CNN are 128^2, 256^2, and 512^2 pixels and the three aspect ratios used are $1:1$, $1:2$, and $2:1$. Before an image is passed through the SSD network it is resized to have the size 300×300 pixels.

For the Faster R-CNN model, one data augmentation strategy is employed. This strategy involves horizontally flipping the image with a probability of 0.5 before it enters the model. For the SSD model one additional data augmentation strategy is employed before the random horizontal flipping. This second strategy involves randomly cropping a part of the original image with a minimum size of 10% of the original image surface and an aspect ratio between 0.5 and 2. This crop is resized to have the size 300×300 pixels and enters the model for training after being horizontally flipped or not.

For the Faster R-CNN model, the learning rate schedule is decided via grid search. Through the grid search we decide to use an initial learning rate of 2.0e-3 and consecutive decreases by a factor ten at the $25{,}000^{\text{th}}$ and $40{,}000^{\text{th}}$ iteration. Justification for decreasing the learning rate at these iteration numbers is provided in the results section. In case of the SSD model, a cyclical learning rate schedule (Smith, 2017) is used, with values ranging between 3.0e-3 and 4.0e-4. Cyclical learning rates are discussed in more depth directly after this paragraph. A more detailed account of the learning rate tuning is provided in the results section.

For both models, stochastic gradient descent with momentum and gradient clipping is used. For the plain vanilla version of gradient descent, the vector that points in the direction along which the weights are updated, the update vector, is the same as the gradient. For gradient descent with

[6] https://github.com/tensorflow/models/tree/master/research/object_detection

momentum, the update vector of the current step is calculated by adding together the gradient, and a momentum parameter times the update vector of the previous iteration. The momentum parameter we use is 0.9. Moreover, gradient clipping at 10 is used, which means that if the L_2 norm of the update vector exceeds 10 then we scale the update vector such that its L_2 norm is equal to 10.

Cyclical Learning Rates

Finding the right learning rate can be a cumbersome process and it usually involves some type of grid search. Smith (2017) introduces an approach, called cyclical learning rates, that does not require experimenting with many different learning rates and schedules. The approach involves increasing and decreasing the learning rate multiple times throughout training. The idea behind periodically increasing the learning rate is to avoid getting stuck in an undesirable local minimum. By taking several larger steps one local minimum can be left behind after which a new and hopefully better local minimum can be reached. Following this line of reasoning, employing a cyclical learning rate schedule can potentially lead to better results than employing a more traditional stepwise decreasing learning rate schedule.

Although the learning rate changes many times during training moving up and down, the learning rate stays within a fixed range. To decide upon the bounds of the range, a model is trained while increasing the learning rate. For the lowest learning rate the parameter updates are typically so small that the model is not learning. In other words, the model accuracy on the validation set is not increasing or is increasing with a negligible amount. At a certain higher learning rate the model will actually start learning. By saying the model is learning we mean that parameter updates generally result in a meaningful increase in terms of validation set accuracy. The learning rate at which the model starts learning is set as the lower bound. At one point the learning rate is too high and the accuracy stops increasing or starts decreasing. The upper bound of the range is set close to this point.

While oscillating between the lower and upper bound of the range throughout the training process, the learning rate should be closer to the upper bound of the range at the beginning of training and at later stages of the training process, the learning rate should be closer to the lower bound of the range. Smith (2017) states that the results of his experiments show that the learning outcomes are robust against cycle length. So, the frequency with which the learning rate is increased and decreased throughout training can, to some extent, be chosen arbitrarily.

3.6 Occlusion Experiments

The idea of the occlusion experiment is to slide a squared grey occlusion box over the image and record how the inference of the model changes. The grey box is supposed to remove all the information in a certain image patch. To obtain the color of the box, the per channel average pixel

intensity is computed over all training images. Each of the images uses RGB color values. The resulting average color is $(R, G, B) = (163, 157, 152)$.

To indicate the size of a squared occlusion box, we use a number between zero and one, s. To get from s to the actual size of an edge of the box in terms of the number of pixels (l), the square root of the total amount of pixels in the image is multiplied by s. In other words, if the image is squared, there would fit exactly s^{-1} occlusion boxes next to each other in the image. The second parameter we specify before conducting an occlusion experiment is the stride r, which is also a number between zero and one. This parameter indicates the size of the steps with which the occlusion box slides over the image. For most of the experiments done here, the stride is chosen to be $r = 0.5$. This means that after occluding the top-left corner of the image, the box is slid $r \times l$ pixels to the right, where l is the length of an edge of the box in terms of pixels. When the right edge of the image is reached, the same sequence of occlusions is repeated but then $r \times l$ pixels lower. So when choosing $r = 0.5$, each pixel in the image is occluded exactly four times over the course of an occlusion experiment. The effect of occlusion for each pixel is averaged over the four occlusion instances.

The effect of occlusion is visualized using heat maps that overlay the original image. Separate heat maps are generated for the effect on the localization accuracy and for the effect on the classification confidence. The localization accuracy is measured in terms of intersection-over-union (IoU). The resulting heat maps consist of squared tiles with the same color intensity. Considering our particular stride ($r = 0.5$), the size of four of these tiles together equals the size of the occlusion box. If a lower stride is chosen, a higher resolution heat map shall be obtained. For one of the experiments we have set $r = 0.1$, which yields a sharp visualization. However, doing an occlusion experiment with $r = 0.1$ requires 25 times as many inferences as with $r = 0.5$.

The colors of the heat map range from dark blue to white to dark red. Red corresponds to a decreased accuracy due to occlusion and blue corresponds a negative decrease or increase of the accuracy due to occlusion. The darker the color, the higher the decrease or increase of the localization accuracy or classification confidence. White in the heat map indicates that the accuracy is unaffected when this part of the image is occluded.

Some images display multiple pistols and in some cases a model might output more detections than that there are actual pistols in an image. In such situations, conducting an occlusion experiment is less straightforward. The way we approach such a situation is to only focus on one handgun at a time and always consider the best detection in terms of IoU. If there are two or more handguns in a single image, then during the occlusion experiment the effect of occlusion on the detection accuracy of only one of these handguns is recorded. Considering the best detection in terms of IoU implies disregarding all other detections output by the model. So, a spurious detection in an unrelated region of the image does not affect the outcome of the occlusion experiments and does not affect the colors of the heat map.

The heat maps can be interpreted as follows. Image patches for which the heat map color is

white can be considered non-informative for the model. That is, if occluding a patch does not influence the quality of the detection at all, then it is reasonable to assume that the model does not use the information in this image patch. Image patches for which the heat map color is a light or dark shade of red are informative and useful for the model. The detection is based on the information in these patches. Finally, patches for which the heat map color is a light or dark shade of blue are counter-informative to the model. It is clear that the model uses the information contained in these image patches, otherwise the quality of the detection would not be influenced by its occlusion. However, the model does not use the information contained in these image patches in the right way. Namely, if the information is taken away the model performs better. This misused information can either be relevant, in which case the model is failing to take advantage of it, or the information is not relevant, in which case it represents noise to the model.

It is worth noting that the interpretation of the heat maps presented above is a simplification and the conclusions are approximate. The colors of the heat map do not conclusively indicate the importance of each pixel. In some rare cases, moving the occlusion box a few pixels can change an inference outcome. However, the heat maps can certainly provide valuable insights as we shall elaborate in the next section.

Next to yielding heat maps for visual interpretation, the occlusion experiments also allow us to compute metrics that quantify what information the model uses. The metrics we introduce are the misapprehension quotient (MAQ) and the tendency to incorporate off-target information (TOffI). In the remaining part of this section these measures are defined.

MAQ

The misapprehension quotient (MAQ) is a measure for the amount of information that the model misuses or "misunderstands". Each part of the image with a blue heat map color is essentially misunderstood by the model. The MAQ is computed by simply dividing the number of pixels for which the performance decrease due to occlusion is smaller than $-\varepsilon$ units, by the total number of pixels in the image. Two MAQs are computed separately for the inference of a model on a single image: one MAQ is computed to measure the percentage of misused information for localization and one MAQ is computed to measure the percentage of misused information for classification.

TOffI

At inference time, object detection methods do not (always) solely use the information that is contained by the pixels inside a small margin around the ground truth bounding box. Irrelevant patches in the image can be a source of noise to the model, or hypothetically some models might even learn to recognize contextual clues and use these clues to detect the object of interest. We introduce a metric that measures the extent to which a model uses and misuses information that is not contained by the pixels in the direct vicinity of the object of interest. We call this metric TOffI,

which stands for tendency to use off-target information. The TOffI is calculated by dividing the amount of used and misused off-target information by the total amount of off-target information. We consider a pixel to contain used information if the assigned decrease due to occlusion is lower than $-\varepsilon$ units or higher than ε units. To decide which information is off-target information, we draw a margin around the ground truth bounding box, which is as wide as the edge length of the occlusion box. Every pixel within the margins is considered on-target information and every pixel outside of the margins is considered off-target information.

For the computation of the MAQ as well as the computation of the TOffI, using a smaller stride leads to a better approximation. However, as mentioned earlier using a smaller stride induces extra computational costs. It is worth noting that different MAQ and TOffI values will be obtained if a different occlusion box size is used.

4 Results

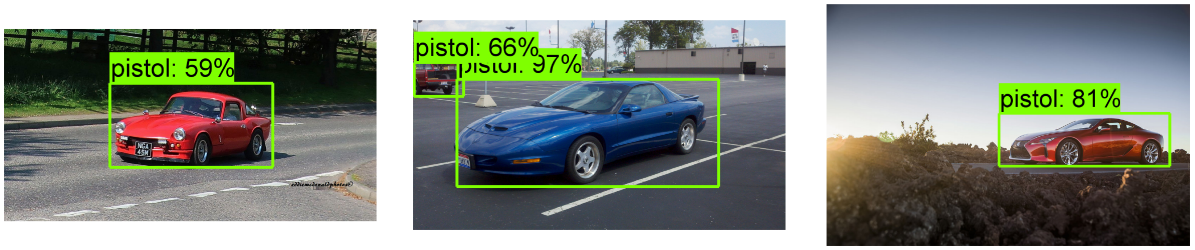
The results section consists of three parts. The first part is devoted to describing important observations related to the training process. This part reflects upon the effectiveness of using cyclical learning rates and discusses the implications of the fact that early versions of the models struggled to distinguish cars from handguns. In the second part, the detection accuracy and speed of the models is evaluated and compared to the performance of the models developed by Olmos et al. (2018). The topic of the third and final part of the results is the interpretation of the detection methods. Here, the research question is addressed, discussing which information is used by the models to come to a detection decision. Several heat maps, which result from the occlusion experiments, are presented, uncovering interesting behavior of the models. Additionally, quantitative measures of information usage are reported.

4.1 Findings Related to Model Training

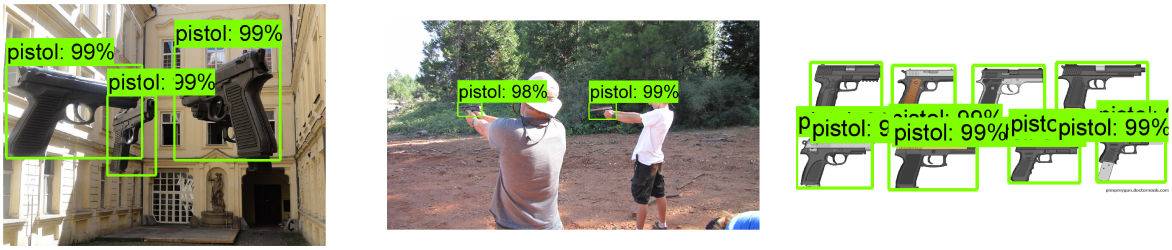
Experienced Complications Associated with Transfer Learning

One of the first Faster R-CNN models that has been fine-tuned for this research project yields undesirable inference results. Namely, if the model is presented with an image that features a car, then the model typically draws a bounding box around the car and assigns the pistol label to it. Figure 8a displays instances in which the model makes such mistakes. Figure 8b illustrates that the model is also capable of detecting handguns adequately.

These inference results suggest that the features that the pre-trained model uses to detect cars are similar to features used by the fine-tuned model to detect pistols. However, the fine-tuned model has not learned which features can be used to distinguish between cars and handguns. A possible reason why the model did not learn to distinguish between cars and handguns is that the training data, which has been used for fine-tuning, contain few images with cars in the background.



(a) Detecting pistols where there are a cars.



(b) Accurately detecting pistols.

Figure 8: Collection of images illustrating that the early stage Faster R-CNN model is capable of detecting pistols, but that it also consistently detects pistols where there are cars.

To obtain a model that ignores cars as being background, the weights and biases in the network need to be updated accordingly. If the training data contain only few images for which cars appear in the background, then such parameter updates do not contribute to minimizing the loss function. Hence, the parameters are not updated in such a way that the model is able to distinguish between cars and pistols. It follows that the model can be flawed, as discussed earlier, despite fine-tuning.

The fact that the above described problem occurs highlights the importance of interpretability and the importance of making careful choices with regards to training data. Such a serious flaw would be much easier to diagnose and avoid in case it were possible to interpret why a model comes to a certain detection decision. A model can only be as good as the data it has been trained on and transfer learning complicates the task of choosing appropriate training data even further.

The Faster R-CNN model that demonstrates the undesirable behavior at inference time has been fine-tuned to detect handguns using exactly the same training data as was used by Olmos et al. (2018) to fine-tune their Faster R-CNN model. However, in their paper the above mentioned problem is not reported. The differences between the two models are that the model we fine-tune is pre-trained on the COCO dataset and Inception-v2 is used as the feature extractor, whereas the model fine-tuned by Olmos et al. (2018) is pre-trained on the ImageNet dataset and VGG-16 is used as the feature extractor. So, it appears that the flawed inference displayed by the model is related to the pre-training. Presumably, the pre-training data play an important role.

To mitigate the problem, annotated images of cars have been added to the training data. From here onwards, this new training dataset will be referred to as the extended training data, as opposed to the old training dataset which will be referred to as the limited training data. The models that have been developed in a later stage of the research process have been trained to explicitly distinguish between cars and handguns. So, if there appears an instance of a car in an image, these models ought to draw a bounding box around the car and ought to assign the car label to such a box. As will be discussed in the next subsection, the solution works as intended. The revised models do not suffer from the same flaws and for our purposes these models suffice. For other purposes, it might be necessary to investigate if next to cars there are other object categories that cause complications.

Effectiveness of Cyclical Learning Rates

To evaluate the effectiveness of using cyclical learning rates (Smith, 2017), we compare the validation set performance of Faster R-CNN models trained using a traditional learning rate schedule with the performance of Faster R-CNN models trained using a cyclical learning rate schedule. The traditional learning rate schedule we adopt is similar to the one used by Huang et al. (2017) to train their model, which is used here to initialize the parameters before training. Huang et al. (2017) use an initial learning rate of 2×10^{-4} for the first 900,000 iterations, until the 1,200,000$^{\text{th}}$ iteration a learning rate of 2×10^{-5} is used, and for the remaining iterations the learning rate is set to 2×10^{-6}. For tuning, we train eight different models trying the eight different initial learning rates of set $L = \{2 \times 10^{-1}, 2 \times 10^{-2}, \ldots, 2 \times 10^{-7}, 2 \times 10^{-8}\}$. Our models need a lot less iterations to train because we take a transfer learning approach and the dataset we use is much smaller than the one used by Huang et al. (2017). The learning rate is incrementally decreased by a factor 10 twice during training. The iterations after which the learning rate is decreased are iteration 25,000 and iteration 40,000.

These iteration numbers have not been carefully chosen but seem valid judging from the graph in Figure 9. Namely, this graph shows that learning is slowing down before the 25,000$^{\text{th}}$ iteration and precisely when the learning rate is decreased for the first time, the accuracy increases steeply. The second decrease of the learning rate at the 40,000$^{\text{th}}$ iteration does not seem to be doing any harm nor good. Overall, Figure 9 provides no reason to believe that decreasing the learning rate after different iteration numbers would make a big difference. Holding these iteration numbers at which the learning rate is decreased fixed, we compare the performance of the models that are trained with different initial learning rates. We observe for training on the limited dataset as well as for training on the extended dataset that an initial learning rate of 2×10^{-3} yields the best performance on the validation set. Furthermore, we observe that for all training instances, the validation set accuracy has more or less stabilized after 50,000 iterations.

Next, the performed steps associated with adopting a cyclical learning rate schedule are de-

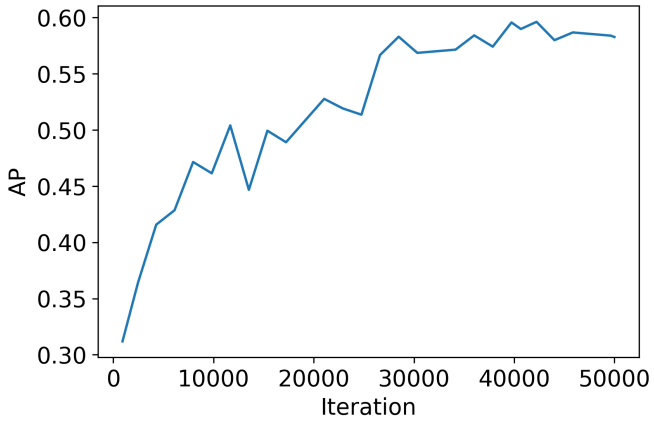


Figure 9: A plot that summarizes how the AP achieved on the validation sample by the Faster R-CNN model changes throughout training with an initial learning rate of 2×10^{-3} and a traditional learning rate schedule.

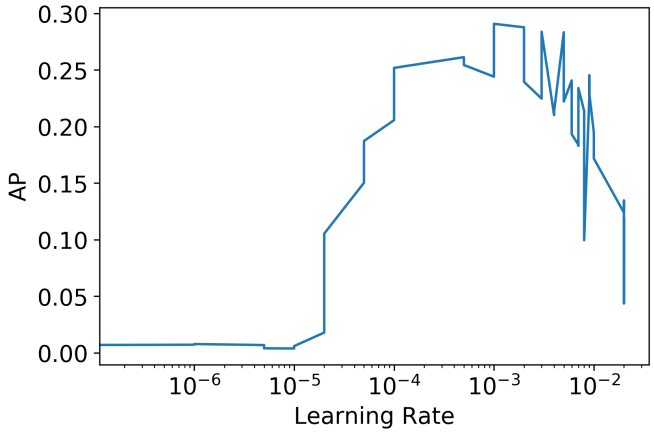
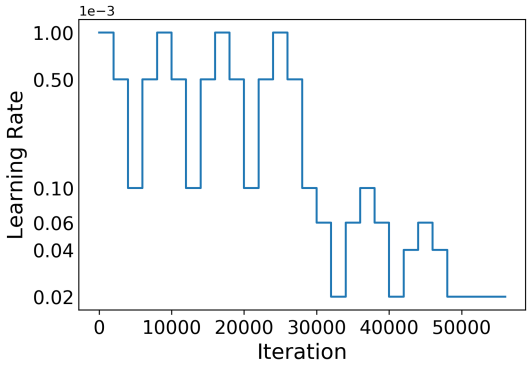
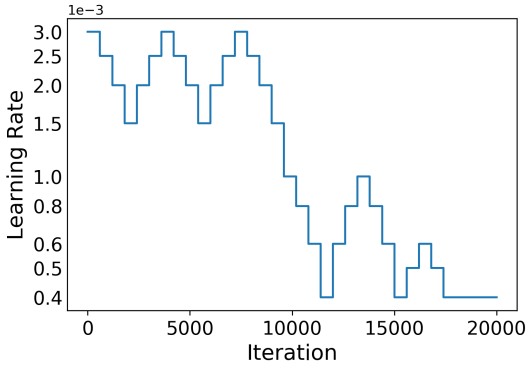


Figure 10: The learning rate plotted against the AP on the validation sample. This plot concerns the Faster R-CNN model trained on the extended training dataset.



(a) A graph that visualizes the learning rate schedule used to train the Faster R-CNN model. The initial learning rate is 10^{-3} and the learning rate at the final iteration, iteration 56000, is 2×10^{-5}



(b) A graph that visualizes the learning rate schedule used to train the SSD model. The initial learning rate is 3×10^{-3} and the learning rate at the final iteration, iteration 20,000, is 4×10^{-4}

Table 1: The performance of Faster R-CNN models trained according to different learning rate schedules.

Metric	Learning rate schedule	
	Traditional	Cyclical
AP	0.57	0.56

scribed. To find an appropriate range for the learning rate, we train a model for a total of 11,000 iterations while increasing the learning rate every 200 iterations. The performance of the model on the validation set at different points of the training process is visualized in the graph of Figure 10. From this graph it can be observed that the AP starts to increase fast at a learning rate of 2×10^{-5} and stops increasing after the learning rate has increased beyond 1×10^{-3}. So, the interval $[2 \times 10^{-5}, 1 \times 10^{-3}]$ is appropriate. The exact learning rate schedule adopted is schematically displayed in Figure 11a. The learning rate changes every 2,000 iterations. Apart from following the general guidelines provided by Smith (2017), the specifics of this schedule have been chosen arbitrarily. This should not be a problem since according to Smith (2017) the training procedure is robust with respect to small changes in the learning rate schedule.

In Table 1 the performance of the Faster R-CNN models described above, in terms of the AP on the validation sample, is reported. The AP achieved by the model that is trained using a cyclical learning rate is 0.01 lower than the AP achieved by the model obtained from doing a grid search. So, using a cyclical learning rate greatly reduces the amount of computation time needed to tune the learning rate and yields a model that performs only slightly worse.

For the interpretative analysis with occlusion experiments, we use the model that shows the best performance on the validation sample, which is the Faster R-CNN model that is trained with a traditional learning rate schedule. The SSD model that we consider in this research project is trained using cyclical learning rates. The learning rate schedule used for training is displayed in

Table 2: Classification performance results on a test set of 608 images. Half of these images feature one or more pistols.

		Extended training dataset		
	Olmos et al. (2018)	Faster R-CNN	SSD	Faster R-CNN
True positive	304	304	304	304
True negative	247	238	204	200
False positive	57	66	100	104
False negative	0	0	0	0

Figure 11b. The learning rate changes every 600 iterations.

4.2 Model Performance Evaluation

Here we evaluate how well the different models perform the task of handgun detection. First the inference results on a test set constructed by Olmos et al. (2018) are reported. Surprisingly, Olmos et al. (2018) let their object detection method perform a classification task on this test set. For the purpose of comparing our models to their model, we perform a classification task as well. An object detection model can be used to do image classification in the following manner. If the model detects one or more pistols, then an image is classified as containing a handgun and otherwise it is not.

The results of the model comparison are displayed in Table 2. From this table it can be seen that for all models the number of false negatives is equal to zero, meaning that all the models are able to identify every single image that contains a handgun. What differs across the models is the number of images in which handguns are detected while there are no handguns present, the number of false positives. For our Faster R-CNN model this number is similar to that of Olmos et al. (2018). Our early generation Faster R-CNN model, which has been trained on the limited dataset, misclassifies most images that contain a car causing the number of false positives to be higher, totalling 104. As expected, the SSD model is less accurate than the Faster R-CNN model. Relative to Faster R-CNN models, SSD models generally sacrifice some accuracy for a large gain in terms of detection speed.

From Table 2 it can be observed that for this set of 608 test images, our best performing model detects a handgun on about one-fifth of the images where there is no handgun present. The SSD model even does so for about one-third of the images without a handgun. Presumably, this high number of false positives can be ascribed to the fact that the images are of very low resolution. All the 608 images have an original size of 160×120.

Next to comparing performance across papers, we report the results related to the cross-validation for estimating the generalization performance. All ten models that have been trained for the cross-validation do not exhibit the previously discussed undesirable behavior. On a set of 79 images, which feature cars but no handguns, the models do not spuriously detect handguns

Table 3: Estimates of the generalization performance of the SSD and Faster R-CNN models obtained through 5-fold cross-validation.

	Fold					
	1	2	3	4	5	Mean
Faster R-CNN	0.631	0.649	0.643	0.640	0.599	0.632
SSD	0.598	0.608	0.622	0.630	0.594	0.610

where there are cars. However, the models do yield some other spurious detections of handguns. These instances in which a model makes a mistake are unrelated to the above described problem with the car class. Each of the SSD and Faster R-CNN models does not yield more than four of such spurious detections on the set of 79 images that feature cars.

Estimates of the generalization performance of the SSD and the Faster R-CNN model are presented in Table 3. We observe that Faster R-CNN performs better than SSD. On each individual fold, a Faster R-CNN model achieves a better AP value. Although for the fifth fold, the performance is very similar for both models. On average the AP achieved by SSD models is 0.022 lower than that achieved by Faster R-CNN models.

To put the performance metrics into perspective, we report the high score on the COCO bounding box detection leaderboard, which is an AP of 0.526 at the time of writing. The APs reported on this leaderboard are an average over multiple classes, whereas the AP reported on our test set only concerns the pistol class. Of course it is important to bear in mind that comparing performance measures across test sets is not very meaningful.

The Faster R-CNN model performs inference in about 0.12 seconds and the SSD model is three times faster with a frame rate of 0.04 seconds. The inference speeds have been measured while using a NVIDIA GPU of the model Tesla K80. Whereas detection is slower for Faster R-CNN, it takes less computation time to train the Faster R-CNN model, given the specifics of our training setup. With a batch size of 16, it takes about 5.5 hours to train the SSD model for all of the 20,000 iterations, which amounts to about one iteration per second. With a batch size of one, it takes about 4.7 hours to train a Faster R-CNN model for all of the 56,000 iterations, which amounts to about 3.3 iterations per second.

4.3 Interpreting the Detection Inferences

In this last subsection of the results we interpret several instances of inferences made by the models. Through these instances we show what kind of interpretations the occlusion experiments can yield and we highlight behavior of the methods that the experiments reveal. Finally, we report several quantitative results from the occlusion experiments.

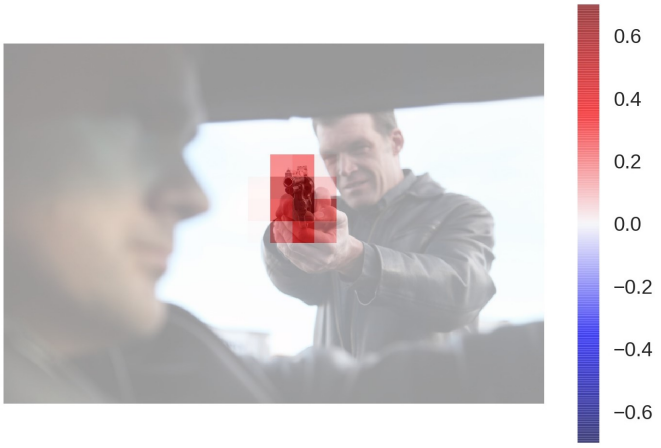


Figure 12: A heat map for the Faster R-CNN model visualizing the decrease in terms of classification confidence due to partial occlusion of the image. The box size and sliding stride are 0.1 and 0.5, respectively.

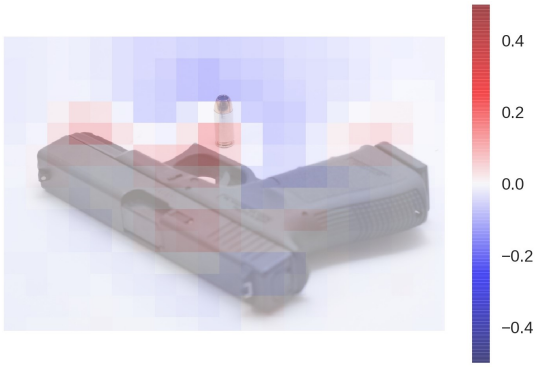
No Surprises

If the object of interest, the pistol, is not occluded, it is desirable that the model detects it with confidence. In case the pistol is (partly) occluded, then it is reasonable to expect a decrease in the classification confidence. This is exactly what we observe for numerous instances. An example is presented in Figure 12. Most parts of the heat map overlaying the image are colored white, corresponding to no decrease in terms of the classification confidence due to occlusion. For the locations in the image where the heat map shows red, it holds that occluding this part of the image decreases the classification confidence. Unsurprisingly, the image patch containing the handgun shows red. Darker red corresponds to a larger decrease, as indicated by the color bar on the side of the image. It is worth noting at this point that the different heat maps throughout this paper have different scales for their color bar.

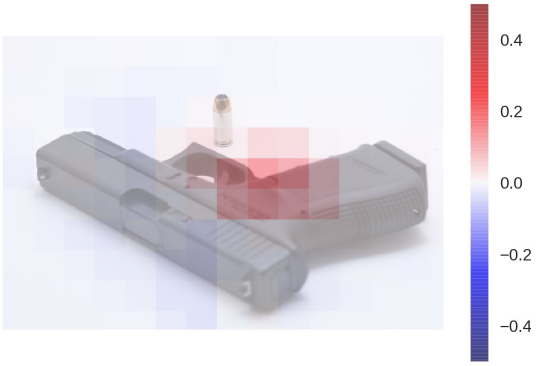
From the heat map of Figure 12 it is clear that the model only uses on-target information for this inference and does not misuse or misapprehend any information. This observation is confirmed by the fact that the MAQ and TOffI corresponding to this inference instance are both equal to zero. These values, and all other MAQ and TOffI values that will follow in this section, have been computed setting $\varepsilon = 0.01$.

Puzzling Localization

Not all occlusion experiments yield such straightforward heat maps as the instance in Figure 12. Here, an image is highlighted for which occlusion influences the localization accuracy of SSD and Faster R-CNN in different ways. It is reasonable to expect that if the edges of a pistol are occluded it will be harder for a model to accurately estimate where to draw the bounds of the box. The first results for the SSD model that are discussed in the next paragraph are in line with this expectation.



(a) The effect of partial occlusion on the localization accuracy of SSD. The box size and sliding stride used are 0.12 and 0.5 respectively.



(b) The effect of partial occlusion on the localization accuracy of Faster R-CNN. The box size and sliding stride used are 0.17 and 0.5 respectively.

Figure 13: Heat maps overlaying an image illustrating what information contributes to correct localization.

However, the results for the Faster R-CNN model are unexpected. In the subsequent example, we observe similar surprising behavior of the SSD model.

In Figure 13a, we see the visualization of an occlusion experiment applied to the SSD predictor. The heat map overlaying the image shows darker red at some (but not all) of the edges of the pistol. The model seems to be especially struggling to recognize where the top edge of the bounding box should be drawn if the muzzle, trigger guard, or bottom of the grip is occluded. So, for this example, the SSD model seems to be using edges of the pistol to decide where to draw the bounds of the box. Whereas, we will observe next that the Faster R-CNN model does not do so for localization on the same image.

Figure 13b shows how occluding different parts of the image influences the localization accuracy of the Faster R-CNN model. From this heat map, it is clear that occluding the edges of the pistol influences the localization accuracy minimally. Only a seemingly arbitrary patch closer to the middle of the pistol shows darker red. So in this case, the Faster R-CNN model is able to find out where the edges of the handgun are, even when the edges are not visible. This is a rather puzzling observation and explaining it is not straightforward. It is possible to infer from the size of the barrel and the orientation of the pistol where the grip of the handgun ends. But, it is unclear if the model does anything like this.

In addition to not looking at the edges of the pistol, the Faster R-CNN does not use off-target information. The TOffI for this inference of the Faster R-CNN model is 0.01, whereas the TOffI for the inference of the SSD model is 0.9. So the SSD model uses 90% of the available off-target image surface and mainly misuses this information. Namely, the heat map of Figure 13a shows lighter and darker tints of blue in the top of the image, which makes up the majority of the off-target image surface. To avoid confusion, the lower right corner does not display any pistol, but for the

Figure 14: A heat map visualizing the effect of occlusion on the localization accuracy of the SSD model. The box size and sliding stride used are 0.12 and 0.5 respectively.

most part it does not count as off-target image surface because it is within the margins around the borders of the ground truth bounding box.

Next, another case is highlighted in which a model looks at a part of the pistol that is close to the middle of the object to decide where to draw the bounds of the box. The heat map corresponding to this case is displayed in Figure 14. For the most part, the heat map overlaying the image shows white. Just left of the case of the watch near the top of the black booklet, the heat maps shows blue, which means that this information is misused by the model. The heat map shows red near the tip of the barrel and the tip of the handle, it shows dark red around the trigger. This dark red color around the trigger means that if the trigger or the area close to it is occluded, then the SSD model is not able to accurately localize the pistol anymore.

The images of Figure 15 confirm that the localization accuracy deteriorates drastically due to occlusion of the trigger. Figure 15a shows that the SSD model is able to localize the pistol reasonably well in the original image, achieving an IoU of 0.85. In Figure 15b, the detection is visualized for the image with a partially occluded trigger. Although the grey occlusion box is located close to the middle of the red ground truth bounding box, the effect of occlusion on the localization accuracy is large. The IoU achieved for the image with occlusion is only 0.65. It appears to be the case that the SSD model uses information contained by image pixels in the neighborhood of the trigger to decide where to draw the bottom edge of the bounding box.

Identifying Unusual Object Attributes

In this subsection, we highlight the images on which the SSD model only partly recognizes the handgun. Partial recognition is meant in the sense that the visual appearance of one part of the pistol prohibits confident detection.

One heat map corresponding to the image under consideration is displayed in Figure 16a. This heat map is colored white for the majority of the image surface, including the chest and the face of

(a) Visualization of the detection of the SSD model in the original image.

(b) Visualization of the detection of the SSD model in the image that contains an occlusion box.

Figure 15: Visualizing the effect of occluding a part close to the middle of the handgun on the localization accuracy achieved by the SSD model. The red box corresponds to the ground truth bounding box.

(a) The effect of partial occlusion on the classification confidence.

(b) The effect of partial occlusion on the localization accuracy.

Figure 16: High resolution heat maps overlaying an image for the purpose of interpreting the inferences made by the SSD model. The box size and sliding stride used are 0.1 and 0.1 respectively.

the person. So, the confidence with which the model is classifying the detected object as a pistol, is not affected when parts of the face or the chest of the person are occluded. However, when the part of the handgun just above the trigger is occluded, the classification confidence drops about seven to ten percent. Correspondingly, this patch of the image shows red on the heat map of Figure 16a.

The noteworthy observation about this heat map is that the part of the barrel just before the muzzle of the pistol shows blue. This means that, if the above mentioned part of the barrel is occluded, the model is more confident about its classification decision. In other words, the visual appearance of the part of the the barrel just before the muzzle of the handgun prohibits confident detection. A possible explanation could be that the pistol in this image has an unusual barrel. The data that the model has been trained on mainly contain instances of handguns with

(a) Detection on the original image.

(b) More confident detection on the image with partly occluded pistol.

Figure 17: Visualizations of the detections yielded by the SSD model with and without occlusion box. The red box corresponds to the ground truth bounding box.

more conventional kinds of barrels. So, possibly the model yields a slightly more conservative classification score in the original image because a part of the pistol looks unfamiliar.

Figure 17 shows that the classification confidence increases when the unusual part of the barrel is occluded. Figure 17a demonstrates the detection on the original image and Figure 17b depicts the more confident detection with an occlusion box. The classification confidence is 87% in the original image and 94% when the part of the barrel just before the muzzle is occluded.

We also observe from Figure 17 that the localization accuracy in terms of IoU deteriorates when the non-typical part of the barrel is occluded. This corresponds to what the heat map of Figure 16b indicates. Namely, on this heat map the unusual part of the barrel shows red. In fact, all colored parts of this heat map show red resulting in a MAQ for localization of 0. However, the MAQ for classification is 0.13.

Investigating a Failure

The final image under consideration is one on which the SSD model is not able to detect the pistol. Why it is not able to detect the pistol is of course not known, and likewise, it is not known if the model is close to detecting the pistol or if it is completely oblivious to its presence. However, it turns out that through an occlusion experiment one can learn quite a lot concerning these two unknowns.

Figure 18 predominantly shows white. This corresponds to a decrease in classification confidence of zero. So, in the original image the SSD model does not detect the handgun and for most parts of the image, occlusion does not make a difference either – still no detection. However, some parts of the heat map show blue, which corresponds to a negative decrease in classification confidence. In other words, if specific parts of the image are occluded, the model starts noticing the pistol. Some of the patches of the heat map showing blue can be deemed uninformative back-

Figure 18: A heat map for the SSD model visualizing the decrease in terms of classification confidence due to partial occlusion of the image. The box size and sliding stride are 0.12 and 0.5 respectively.

ground, in the sense that humans do not use these parts of the image to identify the handgun. But, the information contained in these image patches does influence the SSD model's inference.

Judging from the heat map, a high MAQ and TOffI can be expected. All information that is used by the model is misused by it resulting in a MAQ of 0.17. The TOffI is 0.09, which might seem too low. However, it is important to bear in mind that there is a margin around the ground truth bounding box that is as thick as the edge length of the occlusion box. Only the pixels outside of this margin are considered off-target.

In Figure 19, we illustrate how the SSD model starts noticing the pistol when a part of the background is occluded. Figure 19a shows that there is no detection in the original image and Figure 19b shows that the model catches a glimpse of the pistol yielding a hesitant detection.

What we learned from this occlusion experiment concerning the unknowns at the beginning of this subsection is that the model is actually pretty close to noticing the pistol, were it not for the interference of background noise. So, occlusion experiments can help to understand better what the detection model is doing and how it is influenced. But this example also gives a more general insight. Namely, the inference made by the detection models can be influenced by seemingly unrelated background.

Quantitative Outcomes of Occlusion Experiments

In order to obtain an interpretable model comparison of a quantitative nature, occlusion experiments are conducted on a selection of images from the test set and the corresponding MAQ and TOffI values are computed. A total of 24 images has been selected and all of these images feature a person holding a pistol. The stride and occlusion box size used for the experiments are 0.5 and 0.12, respectively.

The results are summarized in Table 4. In this table, the mean and median MAQ and TOffI

(a) The original image in which no pistol is detected.

(b) Image including occlusion of background on which a pistol is detected with low confidence.

Figure 19: Visualizations of the detections yielded by the SSD model with and without occlusion box. The red box corresponds to the ground truth bounding box.

are reported for the classification and localization inferences of the two models. We report the median in addition to the mean, because a single high observation can drive up the value of the mean substantially.

From the four values in the upper left corner of Table 4 it can be observed that in the context of localization the misapprehension quotient is similar for SSD and Faster R-CNN. The mean MAQ of Faster R-CNN is somewhat higher and the median MAQ is somewhat lower than the mean and median MAQ of SSD. So overall, the percentage of information in the images that is misused for localization purposes is similar for both models. However, the usage of off-target information differs across the models. Namely, when the Faster R-CNN model uses relatively much off-target information it often uses more than the SSD model. However, when the SSD model uses relatively little off-target information it often uses more than the Faster R-CNN model. This can be concluded from the fact that the mean TOffI is almost the same for both models, the fact that the gap between the mean and the median is small for SSD, and the fact that the median is much lower than the mean for Faster R-CNN.

In the context of classification, the Faster R-CNN model generally uses less off-target information and misuses less information than the SSD model. The median MAQ and TOffI for Faster R-CNN are equal to 0. This means that for more than half of the images in the selection the Faster R-CNN model does not misuse information and does not use off-target information for classification. It is no coincidence that the SSD model uses more off-target information and at the same time misuses more information for classification. Namely, off-target information is more often used in an unproductive fashion than in a productive one.

5 Conclusion and Discussion

In summary, this paper has addressed the problem that the inferences provided by object detection models based on CNNs, are difficult to interpret. We have applied the occlusion experiment

Table 4: The quantitative results corresponding to a series of occlusion experiments performed on a selection of 24 images from the test set.

		Localization		Classification	
		SSD	Faster R-CNN	SSD	Faster R-CNN
MAQ	Mean	0.061	0.079	0.064	0.046
	Median	0.037	0.030	0.057	0.000
TOffI	Mean	0.068	0.069	0.099	0.046
	Median	0.066	0.015	0.010	0.000

technique in a new context and discovered surprising behavior of the SSD and Faster R-CNN object detection methods. Namely, it turns out that the models do not always look at the edge of an object to decide where to draw the bounds of a detection box. Furthermore, we show that conducting an occlusion experiment can yield valuable interpretations. For example, we show that it can contribute to an interpretation of why a model fails.

The interpretative analysis has been performed on handgun detection models. We partly replicate and extend earlier work on handgun detection (Olmos et al., 2018). The generalization performance of the detection models has been accurately estimated through five-fold cross-validation. As expected the Faster R-CNN model outperforms the SSD model in terms of accuracy. On the other hand, the SSD model performs inference in one third of the time in which the Faster R-CNN model performs inference.

Through conducting occlusion experiments it is possible to gain insight on what information pistol detection models base their inference results on. Using heat maps it is visualized what parts of the image surface contribute to accurate detection and what parts of the image surface prohibit accurate and confident detection. We introduce a metric, the misapprehension quotient (MAQ), which quantifies the misusage of information by the model. For the experiments that are presented in this thesis, we observe that both models misapprehend about equal amounts of information for the purpose of localization. But, for the purpose of classification the SSD model is using more information incorrectly. The second metric we introduce is the TOffI, which quantifies the tendency to use off-target information. We observe that the Faster R-CNN model tends to use less off-target information when it comes to classification. However, the comparison of the observed TOffI values is ambiguous when it comes to localization.

Our efforts represent a step forward towards interpretable object detection. A direction for further research could be using other interpretation approaches in parallel with the occlusion experiments. As a result, one might be able to fully explain the noticeable observations pinpointed in the current paper. Overall, the results of this paper give reason to be confident that research into finding ways to interpret the inferences resulting from deep learning based methods is worth pursuing.

Finally, we point out several limitations of the interpretation technique used in this paper and

the metrics that are introduced. One downside of the occlusion experiment technique in its current form is that it cannot be used to interpret instances in which a model spuriously detects a handgun where there is no handgun. The technique can be used to find out which information a model uses to come to a correct detection or what information was prohibiting correct detection. However, occlusion experiments cannot be used to find out what information the model uses to come to a spurious detection. Another downside of the technique is that occlusion has an unwanted side effect. Namely, placing an occlusion box in the image does not only take away the information from a specific image patch but also introduces new edges in the image. The detection networks are trained to recognize edges and these new edges could be interfering.

A problem of the MAQ metric is that there is a risk that its value is influenced by the scale at which the handgun is displayed. If a pistol is displayed at a high scale, say the pistol occupies 90% of the image surface, then most information on the image is relevant. Lets zoom out on the same scene, adding more background, until the pistol occupies only 10 percent of the image surface. If the model only uses information close to the pistol, misuses some of that information, and ignores the background, then the MAQ decreases if more background is added. So, in some situations it could be possible to decrease the MAQ simply by zooming out on the scene. Therefore, the absolute value of the MAQ is not too meaningful. Though, it is still meaningful to compare the MAQ values for a certain (set of) image(s) across detection models.

References

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, *88*(2), 303-338. doi: 10.1007/s11263-009-0275-4

Girshick, R. (2015, December). Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (pp. 1440–1448). doi: 10.1109/ICCV.2015.169

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014, June). Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision an Pattern Recognition (CVPR)* (p. 580-587). doi: 10.1109/CVPR.2014.81

Glorot, X., Bordes, A., & Bengio, Y. (2011, June). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (p. 315-323).

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *Elements of statistical learning* (Vol. 2). New York: Springer Series in Statistics.

He, K., Zhang, X., Ren, S., & Sun, J. (2016, June). Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 770-778). doi: 10.1109/CVPR.2016.90

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... Murphy, K. (2017, July). Speed/Accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 3296-3297). doi: 10.1109/CVPR.2017.351

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS)* (pp. 1097–1105). Curran Associates, Inc.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 21–37). Cham: Springer International Publishing. doi: 10.1007/978-3-319-46448-0_2

Long, J., Shelhamer, E., & Darrell, T. (2016, May). Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 3431-3440). doi: 10.1109/TPAMI.2016.2572683

Olmos, R., Tabik, S., & Herrera, F. (2018, January). Automatic handgun detection alarm in videos using deep learning. *Neurocomputing*, *275*, 66–72. doi: 10.1016/j.neucom.2017.05.012

Raschka, S. (2018). Model evaluation, model selection, and algorithm selection in machine learning. *arXiv e-print*, arXiv: 1811.12808. (Accessed on March 18$^{\text{th}}$ 2019)

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016, June). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and*

Pattern Recognition (CVPR) (pp. 779–788). doi: 10.1007/978-3-319-10590-1_53

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28 (NIPS)* (pp. 91–99). Curran Associates, Inc.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013, Dec). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv e-prints*, arXiv:1312.6034. (Accessed on April 3rd 2019)

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv e-print*, arXiv:1409.1556v6. (Accessed on November 4th 2018)

Smith, L. N. (2017, March). Cyclical learning rates for training neural networks. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)* (p. 464-472). doi: 10.1109/WACV.2017.58

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *arXiv e-prints*, arXiv:1312.6034. (Accessed on April 15th 2019)

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 818–833). Cham: Springer International Publishing. doi: 10.1007/978-3-319-10590-1_53