Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Deep Learning Based Robot Detection in the Context of RoboCup

Bachelor Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Robin Papke
geboren am: 10.06.1995
geboren in: Berlin

Gutachter/innen: Prof. Dr. Verena Hafner
Prof. Dr. Hans-Dieter Burkhard

eingereicht am: ..... verteidigt am: .....

Abstract

The great success of applications using deep learning has created a new wave of excitement in recent years. Even though the roots of deep learning reach far back into the past, only recent advances in technology have enabled it to become as successful as it is today. It is now of interest to integrate methods of deep learning into different application domains to test their applicability. With that in mind a state-of-the-art object detection method based on deep learning, named Faster R-CNN [25], was tested to detect robots used to play soccer in the RoboCup. The findings show that a deep learning approach for object detection can produce good results in reasonable time, with only little training data and only low hardware requirements. Furthermore the results suggest that this approach may very well be applied to any other object within as well as outside of the RoboCup.

Kurzfassung

Der große Erfolg von Anwendungen, die mit Deep Learning arbeiten, hat in den letzten Jahren eine neue Welle der Begeisterung ausgelöst. Auch wenn die Wurzeln von Deep Learning weit in die Vergangenheit zurückreichen, so haben doch erst die jüngsten technologischen Fortschritte den derzeitigen Erfolg möglich gemacht. Es ist nun von Interesse, Deep Learning in verschiedene Anwendungsbereiche zu integrieren und die Anwendbarkeit dieser Methoden zu testen. Vor diesem Hintergrund wurde eine hochmoderne, auf Deep Learning basierende Objekterkennungsmethode namens Faster R-CNN [25] zur Erkennung von Robotern getestet, die im RoboCup zum Fußballspielen eingesetzt werden. Die Ergebnisse zeigen, dass ein Ansatz basierend auf Deep Learning zur Objekterkennung gute Ergebnisse in vertretbarer Zeit, mit wenig Trainingsdaten und nur geringen Hardware-Anforderungen, erzielen kann. Die Ergebnisse deuten außerdem darauf hin, dass dieser Ansatz ebenfalls gut zur Erkennung anderer Objekte innerhalb und außerhalb des RoboCups angewendet werden kann.

Contents

1 Introduction

In the course of this thesis we will look at an object detection method based on deep learning in the context of RoboCup. RoboCup provides an excellent platform for this, because the detection of objects is an elementary part of the RoboCup. We will begin this introduction with the development of deep learning in recent years and some insight into why it has become so popular lately. Afterwards, we take a look at the RoboCup, its challenges and its goals. Subsequently, we will focus on the objective of this work and briefly explore some alternative methods based on Deep Learning, which are also used for object detection. The last part of this introduction will provide an overview of the topics covered in this work.

1.1 Motivation

Deep Learning has caused a new wave of excitement in recent years due to its ability to solve difficult tasks like speech recognition and image classification with considerably higher precision than some more traditional methods. However, early work in the field of deep learning can be traced back as early as 1980, when Kunihiko Fukushima published his work on the Neocognitron [7], arguably the first deep neural network. Although this and similar work had great potential, deep neural networks could not be sufficiently trained back then so that they lost some of their appeal in the following years. Key challenges in the training of deep networks, such as the gradient vanishing problem (see [13]), difficult hardware requirements and the availability of large datasets had to be solved to make deep learning more viable in practice.

While the algorithmic challenges of the training were solved successively, the hardware-technical breakthrough was achieved by using cost-effective and powerful graphics processing units (GPUs). In addition to their role in the gaming industry, GPUs are an excellent medium for matrix and vector multiplication, which are essential operations for the training of neural networks. NVIDIA Corporation, a leading manufacturer of GPUs, has recognized this potential early on and has implemented measures to provide additional support for artificial intelligence in its GPUs. At last, the lack of available data for the training of neural networks has become mostly obsolete as a side effect of the digital revolution.

In 2012, a team from the University of Toronto was able to secure a place among the best competitors of the ImageNet Large-Scale Visual Recognition Challenge (see [17]) for the first time with a deep learning based approach. Methods based on deep learning have become the standard for many teams of similar competitions ever since. The rapid success has led to a significant increase in the number of new companies such as MetaMind and Nervana Systems, which offer technologies and services in the field of deep learning and artificial intelligence in general. In 2014, TechCrunch, an online news portal for technology-based topics, reported that the US company Google has bought the British start-up company DeepMind Technologies, which specializes in artificial intelligence, for over 500 million euros (see [28]). Facebook also made headlines on TechCrunch when they hired Yann LeCun, professor at New York University and

leading expert in artificial intelligence, as head of their artificial intelligence laboratory (see [3]) in 2013.

It is now of interest to use deep learning in different application settings and evaluate its performance. For this reason, a deep neural network was trained using a state-of-the-art deep learning approach for object detection. The aim of this network is to identify and locate robots of the RoboCup Standard League in images.

1.2 RoboCup

RoboCup is an annual international competition where students and researchers from around the globe can compete in a variety of robotics related challenges. The general aim is to provide researchers and students exciting challenges in the field of artificial intelligence and robotics to promote research in these areas. The competitions encourage teams to constantly try out new ideas and improve on their work so that they can keep up with the other teams. The RoboCup currently consists of multiple soccer leagues, but it also offers challenges unrelated to soccer, such as the rescue robot competition. One of the soccer leagues is the Standard Platform. This league specifies the robots that are allowed to be used for playing. In order to successfully participate in the Standard Platform, participants must provide their robots with the necessary skills to play the game autonomously. This includes running on the field, passing the ball and scoring goals. These tasks can usually be solved in a more efficient way if the robot detects the relevant objects fast and accurate.

1.3 Contribution

The following work evaluates the applicability of a deep learning based method for the detection of robots that are used in the Standard Platform of the RoboCup. In addition to potential improvements for competing in the RoboCup, we may also derive further information about the applicability of this method for other objects and applications. Furthermore, this work provides insights into the theoretical background and resources that are required for the successful application of a deep learning method.

1.4 Related Work

There are many different methods based on deep learning that can be used for object detection. One of these methods is Overfeat [27], which was awarded in 2013 as the winner of the Localisation Challenge of the ImageNet Large Scale Visual Recognition Challenge. Overfeat applies a neuronal convolutional network to several locations in an image over multiple scales to predict classes and bounding boxes containing the objects. The predicted bounding boxes are then accumulated for each location and category, to increase the total detection accuracy. Another method for object detection is the Single Shot MultiBox Detector [20], also called SSD. SSD uses a deep neural network and default bounding boxes of different scales and aspect ratios to detect objects. It predicts scores for the default boxes and adjustments for them to better

match the objects they enclose. Since SSD does not calculate object proposals but uses default boxes, it is much faster than other methods while still achieving good results. Another method for object recognition is an approach called YOLO [24]. Unlike other approaches used to detect objects, YOLO does not use classification to perform the detection. Instead, it regards object detection as a regression problem of spatially separated bounding boxes and their class probabilities. Like SSD, YOLO is faster than many other methods and still achieves good results.

1.5 Structure

The work is divided into five sections. In the subsequent second section, we will discuss some of the basics of machine learning and some more specific concepts of artificial neural networks and deep learning. While the more general part on the fundamental ideas of machine learning provides an introduction to the subject, the deep learning section already includes information that serve as the basis for the later sections. The third section deals in more detail with the object detection method relevant for this work. We begin with a look at the predecessors of Faster R-CNN and how the architecture has evolved before we see how Faster R-CNN itself works. In the fourth section, we will discuss how the method was applied for our particular case and look at the results. In the fifth and final section, we will continue to elaborate on the results and consider what can be improved for future applications.

2 Background

The field of machine learning consists of a variety of different methods such as Decision Tree Learning, Support Vector Machines, Deep Learning and more. Many of these methods share common ideas, so it makes sense to get a general overview of the subject before going into more detail. In the first part of this section we will introduce some general concepts, terms and models used in machine learning. In the second part of this section, we will examine a specific type of neural network, the so-called convolutional neuronal network, and address how artificial neural networks can be trained.

2.1 Machine Learning

Intelligent systems are already affecting the everyday life of many people today. Customers of the streaming platform Netflix, for example, supposedly receive film recommendations based on systems that apply machine learning. The online warehouse Amazon allegedly uses machine learning to generate product recommendations and product combinations to increase their sales. Although some global players have already integrated these intelligent systems into their ecosystems, there are often uncertainties about what machine learning is actually about. Kevin P. Murphy provides a good definition for machine learning. „In particular, we define machine learning as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty (such as planning how to collect more data!)“[23]. For example, machine learning makes it possible for the machine to perform tasks such as image recognition and speech recognition, which are very difficult to accomplish with a manually programmed system, more easily. Moreover, machines can process considerably more data than humans, which potentially allows them to recognize patterns in data that is too complex for humans. For this and other reasons, machine learning has great potential and could become one of the most important areas of computer science in the future.

2.1.1 Learning Algorithms

Before going into detail about specific learning learning algorithms, we will first take a look at what learning in the context of machine learning actually means. Tom M. Mitchell provides an appropriate definition. „A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E“ [22]. Mitchell's definition is essentially made up of the three elements: task, experience and performance. The task can be interpreted as the ability that the machine is to acquire through learning. This could be the ability to detect robots in images, for example. The experience is the data that is processed by the machine during training. The performance is a kind of measurement that assesses the quality with which the machine solves a given task.

In order to learn from data and thereby increase performance at specific tasks, machines make use of learning algorithms. These algorithms essentially extract information from data to generate a model of the underlying data generating process (see [1]). In order to generate this model, the learning algorithm requires an initial model that can be optimized based on the given data. This initial model already incorporates assumptions about the data generating process before the optimization. Without these assumptions, it would not be possible to map a specific data generating process, since there is an infinite amount of processes that possibly could have generated the data. The space of possible solutions restricted by these assumptions is sometimes referred to as the hypothesis space of the model, whereby the trained model represents the final hypothesis. It is also worth mentioning that assumptions made by the initial model impact the efficiency with which the model can be optimized. The efficiency also depends on the choice of the learning algorithm itself, which means that the choice of a suitable learning algorithm and model is closely linked. Due to the wide variety of learning algorithms and models, we will only take a look at some of the common approaches of learning algorithms.

Regression. This approach is a statistical process used to determine the relationship between variables. One specific type of regression analysis is linear regression, which is used to model a linear relationship between a scalar $y \in \mathbb{R}$ and a vector of inputs $x \in \mathbb{R}^n$. Linear regression tries to estimate a function $g(x) = \hat{y}$ for which the difference between the true value y and the value $\hat{y}$, that is predicted by the estimated function, is minimal when given the same input. We define the estimated function as $g(x) = \theta^T x$, where $\theta \in \mathbb{R}$ is a vector of parameters. These parameters control the influence of each input in x on the resulting scalar $\hat{y}$. Therefore, we can map an accurate model of the original relationship by optimizing θ based on the information provided by known input and output variable pairs. Besides linear regression there are other common types of regression analysis such as logistic regression.

Clustering. This approach tries to automatically identify similarities in given data by analyzing existing inherent structures within it. One particular algorithm that uses this approach is k-means. This clustering algorithm takes a set of data points $x_1, x_2, ..., x_n$ and a number of clusters k as input. It then places so called centroids $c_1, c_2, ..., c_k$ at random locations in the vector space of the data points. Next the algorithm assigns each data point x_i to the centroid c_j that has the smallest distance to it using an appropriate distance metric. Each centroid then calculates the mean value of the data points assigned to it and moves to this location. The last two steps are repeated until no data points change their affiliation to centroids.

Bayesian Algorithms. These algorithms optimize models that in some form apply Bayes' theorem, a formula that describes how to calculate conditional probabilities. Let $P(H)$ be the probability of the hypothesis H being true and $P(E)$ the probability of an event happening that is related to the probability of H being true. Using Bayes' theorem the conditional probability of H being true provided that E already has

happened can be calculated as

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}.$$

A particular model that applies this formula is the Bayesian network. A Bayesian network is a graphical representations of conditional probabilities between random variables. The nodes of the network represent random variables and are connected by edges, which represent the probabilistic dependencies between them. Some learning algorithms used for optimization of these networks are made up of two components: a scoring metric and a search method. The scoring metric returns a score that is proportional the posterior probability of the network by using given data, while the search procedure generates further networks for evaluation by the scoring metric (see [12]).

Artificial Neural Network Algorithms. These algorithms are explicitly used to train artificial neural networks (ANN) that are inspired by the neural networks of the brain. An example of such a network can be seen in Figure 1. ANNs are compositions of interconnected neurons, also called units, whose influence on the network is described by the weights and biases of their connections. When the connections of an ANN don't form a cycle, they are considered feedforward networks since information is propagated in one direction. Learning is often done by changing the weights of these units using an optimization strategy. The optimization of ANN's will be discussed in more detail later.

Models trained using learning algorithms can be defined by their underlying purpose as either descriptive or predictive. Descriptive models provide information about big datasets, which might not be directly observable through conventional methods of analysis due to the complexity of the data. Predictive models on the other hand are used to make predictions about data that will be generated in the future. The choice of which particular learning algorithm is best suited to train a specific model is difficult. An obvious solution would be to decide by testing the performance of each algorithm using similar conditions. This is greatly inefficient and therefore no practical solution. Some practitioners decide which algorithm to choose by analyzing the complexity of the task that is to be solved by the machine. Often times it makes more sense to solve a simple task using a simple method than to invest a lot of effort into implementing a much more complicated method. Therefore, a general preference for simplicity among possible solutions may be advantageous.

2.1.2 Learning Approaches

Data is sometimes considered the core element of machine learning, as it is the main source of information. Therefore, one way to categorize learning algorithms is to see, what kind of data the algorithm uses and how the learning system processes feedback. The following describes three possible approaches to categorize the learning process.

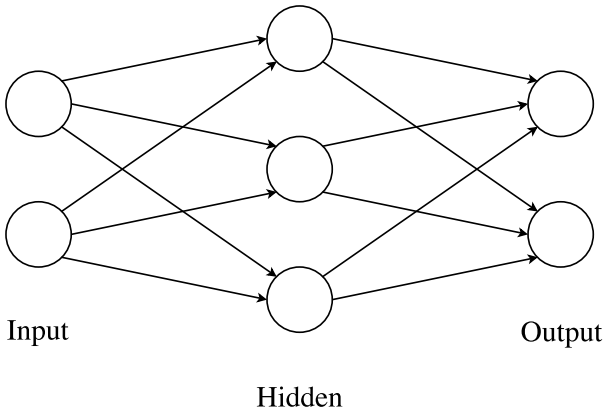


Figure 1: A fully-connected artificial neural network with three layers.

Supervised Learning. These types of learning algorithms learn to associate an input and an output by learning from examples of inputs and predefined outputs. Supervised learning algorithms can furthermore be categorized as classification or regression algorithms. Classification algorithms try to predict results as discrete output, while regression algorithms try to predict results within a continuous output. Although these types of algorithms have delivered excellent results in recent years for tasks such as image recognition, the amount of work involved in predefining outputs for specific inputs is a major drawback compared to other approaches.

Unsupervised Learning. These type of learning algorithms derive structure from data that is unknown prior to learning. Typically the underlying task is to find an optimal representation of some unstructured data that preserves essential information while following predetermined constraints about the structure. Algorithms that extract this structure can furthermore be categorized as either clustering algorithms or dimensionality reduction algorithms. Clustering algorithms group instances of data by similarity of their inherent features. Dimensionality reduction algorithms try to compress information about the given data into a smaller representation.

Reinforcement Learning Unlike other types of learning algorithms, these learning algorithms do not consume fixed data sets. Instead, they learn by actively interacting with an environment. The feedback loop generated by this process enables the machine to learn which actions perform best in specific situations. Figure 2 illustrates this interaction. An obvious disadvantage of this approach is that the environment and its conditions must be definable and of appropriate size in order to learn efficiently. This is perhaps the reason why this approach is preferably used for game environments.

Another way to create this same categorization is to examine the kind of feedback that the learning algorithms receive. Supervised learning algorithms receive feedback through comparison with predefined examples. Unsupervised learning algorithms do not receive any feedback at all and reinforcement learning algorithms receive feedback irregularly from an environment depending on which actions they performed.

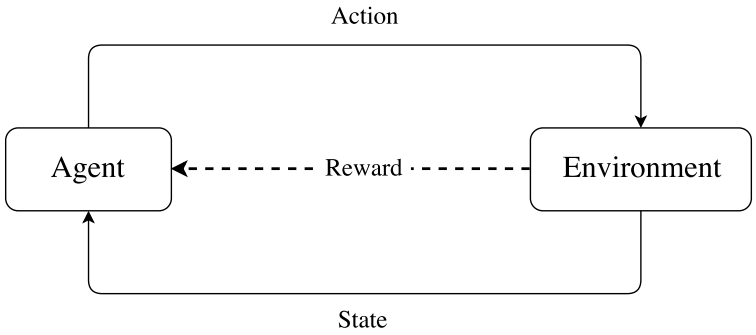


Figure 2: The feedback loop between the agent and the environment.

2.2 Deep Learning

One area of machine learning that has received a lot of attention in recent years is deep learning. The field itself revolves around learning algorithms that are used for training deep neural networks. These are neural networks that contain more than one hidden layer. Recent progress in this area has led to deep neural networks, which have shown a significant increase in performance compared to more traditional machine learning models. Furthermore, deep neural networks have become the state-of-the-art for many machine learning tasks such as image recognition, speech recognition and natural speech processing. In the following section we will examine a specific type of deep neural network and look at how neural networks can be trained.

2.2.1 Convolutional Neural Networks

Deep Learning has produced a variety of different models in recent decades. While the range of application is limited for some of these models, others have been able to show good results for a multitude of problems. Two of these models are recurrent neural networks (RNN) and convolutional neural networks (CNN). RNNs excel at processing sequential data and have achieved state-of-the-art performance on tasks like machine translation [15] and speech recognition [11]. CNNs on the other hand excel at processing data that has a known grid-like topology (see [10]). In particular this includes image data, which may be interpreted as a two dimensional grid of pixels. In the following section, we will only focus on convolutional neural networks because they are the kind of networks typically used for object detection.

Over the last decade, CNNs have not only been able to compete with other machine learning models used for image processing, but also set new records in image processing competitions. One of the first CNNs that received great attention for its performance was AlexNet, a contribution to the 2010 ImageNet Large Scale Visual Recognition Competition [17]. In the following years, more and more participants recognized the potential of CNNs and applied them to different datasets. CNNs today have become state-of-the-art for many image processing tasks. In the following section, we will look at the structure and function of these networks. Since the architecture of CNNs may differ across applications, we will describe the individual stages found in most networks.

The first stage is the convolution stage. Here, each layer is a feature map that represents the occurrence of a single feature across the entire input. The feature itself is defined by the units of the feature map, the kernels. Within a single feature map, each kernel is equal and differs only in its dedicated receptive field. This field defines the specific part of the input that is relevant to the kernel. A kernel contains a number of weights, which describe where the feature would be present within its receptive field. Applying these kernel to the input image is referred to as discrete convolution, which can be mathematically be expressed as

$$(f * g)(n) := \sum_{k \in D} f(k)g(n-k).$$

After each feature map has been calculated, the detector phase begins. Here, the values calculated from the previous stage are normalized by applying activation functions. There is a number of activation functions available, each with slightly different properties. We will focus on one specific activation function which is called rectified linear unit. This activation function changes each negative input value to zero, which simplifies later computations significantly. The output will be a set of feature maps with values equal or above zero.

The third stage is the pooling stage. It summarizes the values of feature maps that are in close proximity to one another. One way to achieve this is by applying max pooling, which determines the maximum value among rectangular selections within the feature map. By reducing the dimension of the feature map we create an invariance to small shifts and distortions of the input (see [18]). This is especially useful if the exact location of a feature is irrelevant compared to its presence. The convolution stage, the detector stage and the pooling stage are sometimes summarized as the convolutional layer of the network.

As mentioned above, each of these stages may appear in a different order, depending on the architecture selected and may occur more than once. Figure 3 shows a example of a simple architecture. Once all feature maps have been calculated, they are passed to the last component that could, for example, classify the results of the previous layers.

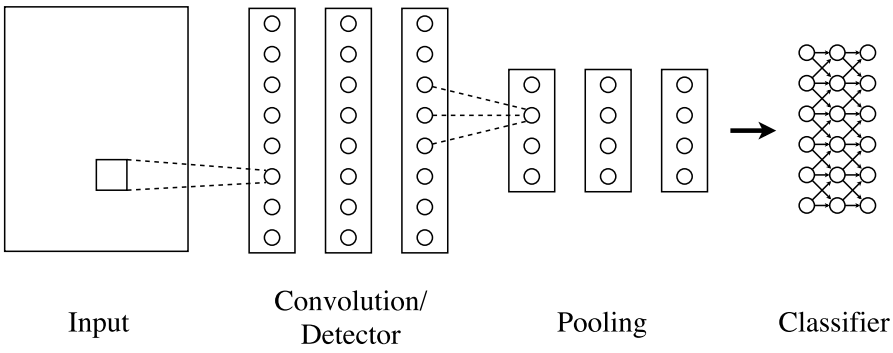


Figure 3: A simple architecture for a convolutional neural network.

2.2.2 Gradient Descent

Learning itself is usually realized as a kind of optimization procedure. Optimization is generally done by either maximizing or minimizing some function $f(x)$ with respect to its input value x. Since maximizing a function $f(x)$ can also be accomplished by minimizing the function $-f(x)$ we will only focus on minimization in this section. An optimization algorithm that is commonly used in machine learning and deep learning is gradient descent. Before going into more detail about gradient descent we must first define what is known as the hypothesis of a model. The hypothesis describes a data

generating process as a function, which is defined as

$$h_\theta(x) = \theta^T x,$$

where $\theta \in \mathbb{R}^n$ is a vector of parameters and x is a vector of input values. Assuming a supervised learning setup, the hypothesis can be trained using a cost function and a set of training examples $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$. Please notice that $x^{(i)}$ is used to describe an element of the training set while x_i is used to describe an element of the input vector x. The cost function estimates the difference between the hypothesis and the data. It does so by calculating the error between the predicted results of $h_\theta(x^{(i)})$ and the true results $y^{(i)}$. One commonly used cost function is the squared error function:

$$J(\theta) := \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2.$$

The goal is to minimize this cost function in order to improve the hypothesis. This is where we can use gradient descent. It analyses how changing a parameter of $J(\theta)$ would change the error by computing its partial derivatives, which we can denote as

$$\frac{\partial}{\partial \theta_j} J(\theta).$$

One way to capture the resulting multitude of partial derivatives of $J(\theta)$ is to summarize them as a vector

$$\nabla_\theta J(\theta) := (\frac{\partial}{\partial \theta_0} J(\theta), ..., \frac{\partial}{\partial \theta_n} J(\theta))^T,$$

which is referred to as the gradient of the cost function with respect to its parameters. Minimal changes to θ to the opposite of $\nabla_\theta J(\theta)$ will move the value of $J(\theta)$ towards its local or even global minimum. The size of the change is controlled by applying a learning rate α to the change. The gradient descent update

$$\theta := \theta - \alpha \nabla_\theta J(\theta)$$

is repeated until $J(\theta)$ converges. Implementations often first compute and store the update of each parameter individually using

$$\hat{\theta}_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}.$$

In this case, the parameter updates still have to be performed at the same time and not individually. This algorithm is also known as batch gradient descent, because it has to accumulate the cost of all training samples in order to perform a single update. This is a huge disadvantage in terms of the computational costs when training with

larger datasets. A faster version of this algorithm, stochastic gradient descent, enables efficient training even on large amounts of data. In contrast to the batch gradient descent, stochastic gradient descent performs an update after each randomly selected training example. However, the approximation of the true gradient introduces noise, which limits the speed at which stochastic gradient descent converges (see [2]). One way to overcome this limitation and decrease the noise, introduced by random sampling, is to gradually decrease the learning rate α after each update (see [10]).

2.2.3 Backpropagation

The flow of information in feedforward networks is called forward propagation. In the last step of this process, the final layer produces an output. Assuming a supervised learning setup we can calculate the difference between the output of our neural network and the true output using a cost function. As mentioned earlier one way to minimize this difference is to compute the gradient of the cost function and use it to change the parameters accordingly. In a deep neural network, however, there is not a simple function to be trained, but a complicated composite function. For a long time there was no method to calculate the gradients for the functions of neural network efficiently, which meant that the hidden units could not be trained precisely enough.

In the late 80s a method called backpropagation (see [26]) solved the issue. The idea is to propagate a signal representing the error through the neural network by using a generalized approach to the chain rule of calculus. The reason why this approach is possible is due to the fact that neural networks at their core are large composite functions. This means that each layer except the input layer, where no computation takes place, can be represented using a composite representation of all previous layers. Once the error signal has been propagated through each unit of the neural network it can then be used to compute the gradients of the cost function with respect to the weights for the units individually. Since the backpropagation process is based on a generalized version of the chain rule of the calculus, it is efficient to use and allows to better model the inner representation of the data generation process.

Before we go into the individual steps of the backpropagation algorithm, we first need to understand the chain rule of calculus. The chain rule of calculus allows us to compute the derivative of a composition of two or more functions. Assuming the reader is familiar with the standard definition, we will move onto the more generalized definition. We define the functions $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^m \to \mathbb{R}^n$. The chain rule of calculus states that

$$\frac{\partial}{\partial x_i} f(g) = \sum_j \frac{\partial}{\partial g_j} f(g) \frac{\partial}{\partial x_i} g_j(x).$$

This enables us to propagate the error signal through the individual layers. To describe the algorithm, we need to define an additional terminology. First, we define a differentiable activation function $act : \mathbb{R} \to \mathbb{R}$, which takes the sum of weighted activation values from the previous layer as input and calculates an activation value as

output. Next we define the vector of all calculated activation values of layer l as $a^{(l)}$. Finally, we will calculate the cost of the computed results and the true results for layer l as

$$E = \frac{1}{2}(a^{(l)} - y^{(l)})^2.$$

We start the backpropagation algorithm by calculating the error signal of the last layer L using the gradient of the cost function with respect to the activation values of the last layer:

$$\delta^{(L)} = \frac{\partial}{\partial_{a^L}} E = a^{(L)} - y^{(L)}.$$

Next, we calculate the gradient of the activation function with respect to the weights of the incoming activation values of the previous layer. Since there is only one weight for each activation value received, there is only one incoming activation value for each partial derivative:

$$\frac{\partial}{\partial_{\theta_i^{(L-1)}}} act = a^{(L-1)}.$$

Then we can calculate the gradient of the cost function with respect to the weights of the incoming activation values by using the chain rule of calculus:

$$\frac{\partial}{\partial_{\theta_i^{(L-1)}}} E = \delta^{(L)} a^{(L-1)}.$$

Finally, we compute the gradient of the cost function with respect to the activation values of the previous layer by multiplying the weights of the previous layer:

$$\delta^{(L-1)} = (\theta_i^{(L-1)})^T \frac{\partial}{\partial_{\theta_i^{(L-1)}}} E.$$

We repeat these steps until we reach the first hidden layer and then use the gradients of each unit with respect to the weights to modify the weights by applying method like gradient descent. By doing so we will receive a deep neural network, which accurately depicts the inner representation of the data generating process and thereby can compete with most traditional machine learning models in terms of performance while still being efficient to compute.

3 Region Based Convolutional Neural Networks

Deep learning alone offers a multitude of methods to perform object detection and has already been able to outperform more traditional systems like the deformable part model [6], which is based on HOG-like features [4]. One of the most successful methods of deep learning for object detection are region based convolutional neural networks. In the following, we will look at the functionality and development of this method in recent years.

3.1 R-CNN

Prior to 2014, the performance of methods for object detection measured with the PASCAL VOC [5] dataset stagnated for a long time. In addition to the MS-COCO [19] dataset from Microsoft and the ILSVRC dataset from ImageNet, PASCAL VOC provides one of the most commonly used datasets for object detection. These datasets contain class descriptions for objects as well as their surrounding coordinates. A paper published in 2014, proposed a new method called regions with convolutional neural network features [9] or R-CNN, which was able to improve the mean average precision (mAP) by more than 30%, relative to the best performing method at that time measured with the PACAL VOC 2012 dataset.

One of the key questions for object detection with deep learning has been how to locate a multitude of objects in images using deep neural networks. In order to locate objects, R-CNNs use region proposal methods like selective search [29] to extract regions from the image beforehand. Afterwards the different-sized regions are cropped and warped into a format that is compatible with the requirements of the CNN. The CNN then produces feature maps of these regions, which are classified using a set of class specific linear support vector machines. Additionally R-CNN trains a linear regression model in parallel that is used to correct the proposed regions.

The second question has been how to train a deep neural network with only a small amount of available training data, because unlike classification, images used to train object detection also need to contain the coordinates of the bounding boxes for each object within an image. Therefore, training data used for object detection is much scarcer than training data used for classification. R-CNN solves this problem through supervised preliminary training followed by domain-specific fine-tuning. The pretraining of the CNN can be done using one of the many large datasets available for classification. It is only necessary to replace the final classification layer of the trained CNN with a randomly initialized classification layer appropriate for the object detection training dataset. During fine-tuning, R-CNN handles region proposals with an intersection-over-union (IoU) overlap with a ground-truth box of over 50% as positives and region proposals below that threshold as negatives examples for the class of that box. IoU can be computed by dividing the area of overlap between the bounding boxes by the area of union. The fine-tuning of the regions is done using stochastic gradient descent. Finally, to train the support vector machines, R-CNN defines an IoU overlap threshold of about 30% that defines when a class is positive for a given

region. This has to be done for each SVM individually. Figure 4 shows the architecture of R-CNN. R-CNN first warps the region proposals into a format that is compatible with the input of the CNN. The warped regions are then feed-forwarded through the convolutional layer and the fully-connected layer of the CNN. The extracted features are evaluated per class using support vector machines and the regions are corrected using the regression layer.

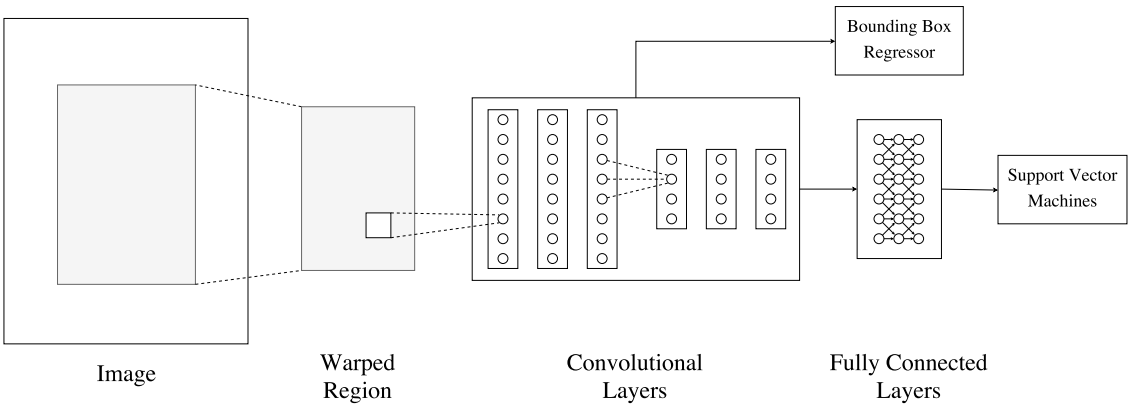


Figure 4: An illustration of R-CNN processing an image.

3.2 Fast R-CNN

Although R-CNN has shown great improvements in the area of object detection, there are some problems regarding this method that have not been mentioned yet. One problem is the rather complicated multi-stage training pipeline, in which a CNN, several support vector machines and a linear regression model are trained in different stages. Another problem is that the extracted features have to be cached on a hard disk. This not only requires hundreds of gigabytes of storage space, but also costs additional training time. A third disadvantage is that the test time of R-CNN is very long, as it has to go through a complete CNN run for each regional proposal. One year after the publication of R-CNN, a new method called Fast R-CNN [8] was introduced, which addresses these problems.

In contrary to R-CNN, Fast R-CNN swaps the order in which region extraction and feature extraction are performed. First, it takes a high-resolution image and feeds it through the convolutional layers of the CNN. Afterwards a so called region of interest (RoI) pooling layer extracts features from the high-resolution feature maps of the last convolutional layer using regions that were proposed for the original image input. At last, the computed features for the given regions are fed into fully-connected layers, which end in a classification head that classifies the objects, and a regression head that corrects the boundary boxes. Figure 5 gives an overview of the architecture of Fast R-CNN. This approach resolves some of the problems we have mentioned earlier. Since

the convolutional features are now shared for all region proposals, Fast R-CNN has a much shorter test-time compared to R-CNN. Additionally the replacement of the multi-stage training pipeline with a single-stage training pipeline now allows the use of backpropagation and therefore reduces the training-time significantly. It is also no longer necessary to store large amounts of feature data to disk.

An important innovation in [8] is the RoI pooling layer. In R-CNN the problem was that although the input of the CNN can be of different sizes due to its scale invariance, the input of the fully-connected layers is always expected to be of low-resolution. Fast R-CNN solves this by projecting the region proposals onto the feature maps of the last convolutional layer and dividing it into a grid of sub-windows. Afterwards it applies max pooling on each sub-window to obtain fixed-sized low-resolution feature maps, which can be fed to the fully-connected layers. Since the whole idea is based on max pooling, we can use backpropagation to train these layers just like any other CNN and save training-time. The network is optimized using stochastic gradient descent.

Using a pre-trained VGG-16 CNN on the Pascal VOC 2007 dataset Fast R-CNN was 8.8× faster to train and 146× faster to test compared to R-CNN (per image) without including the time to compute region proposals (see [8]).

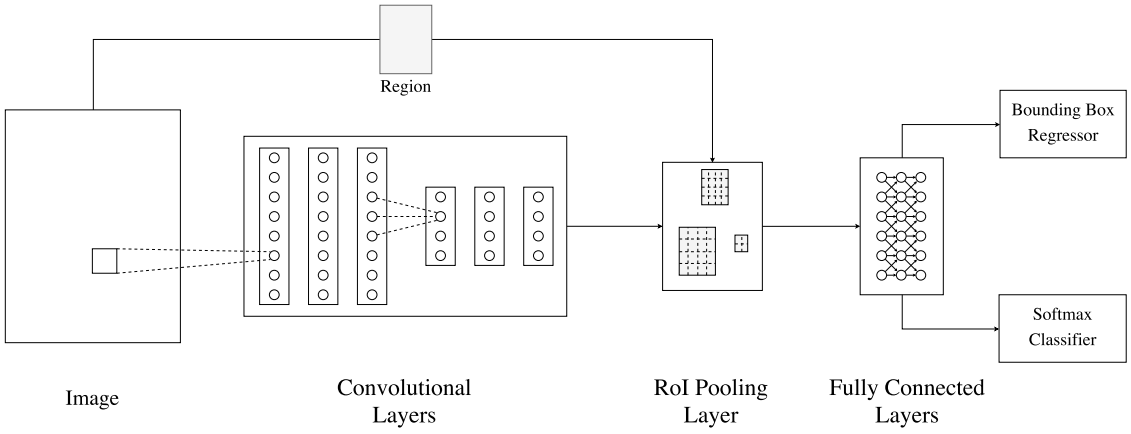


Figure 5: An illustration of Fast R-CNN processing an image.

3.3 Faster R-CNN

The outstanding improvement of Fast R-CNN's test-time with respect to earlier work exposed a new bottleneck: the time to compute region proposals. Even fast region proposal methods like EdgeBoxes [31] still take just as much time to compute region proposals as Fast R-CNN needs to detect objects. A method addressing this issue is called Faster R-CNN [25], which was proposed in the same year as Fast R-CNN. Faster R-CNN introduces so called region proposal networks (RPN), which can be used to generate region proposals for the detector network of Fast R-CNN using the same convolutional features. Detector network of Fast R-CNN hereby refers to the

stages after the convolutional layers. Using this approach the researchers were able to decrease the cost of proposing regions significantly.

The RPN itself is a small fully-convolutional network (see [21]) that processes images of variable sizes and outputs rectangular shaped region proposals for them. It contains a $n \times n$ convolutional layer as well as an 1×1 convolutional layer for box-regression and an 1×1 convolutional layer for object-classification. The box-regression layer produces 4 coordinates, whereas the object-classification layer computes the probability for the spatial windows being either an object or a part of the background, which is expressed by what is called the objectness-score. To combine the RPN and the detector network, Faster R-CNN slides the RPN over the feature maps of the last shared convolutional layer. After each shift the $n \times n$ spatial window is processed by the $n \times n$ convolutional layer of the RPN and the results are fed into the box-regression layer as well as the object-classification layer respectively. Notice that the parameters of the RPN therefore are shared across all spatial windows of the feature maps. Figure 6 illustrates this process.

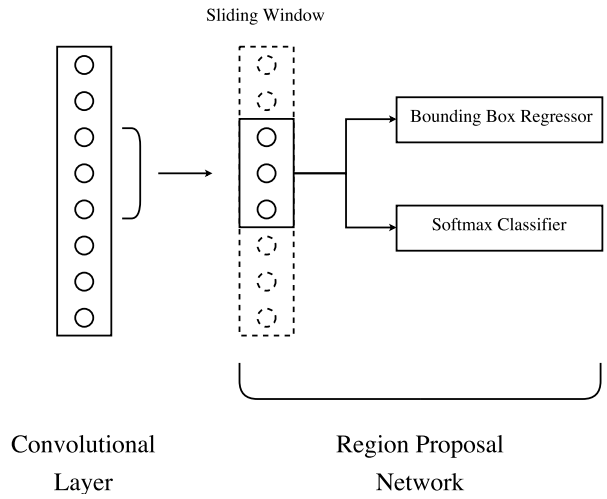


Figure 6: An illustration of how the RPN slides over the last convolutional layer

Instead of using the spatial window itself as the input for the RPN, Faster R-CNN generates k region proposals using so called anchor boxes. An anchor is located at the center of each spatial window at each sliding position and is associated with a scale and an aspect ration. This means that the box-regression layer produces $4k$ outputs and the object-classification layer produces $2k$ outputs per spatial window. The anchor boxes not only accommodate different types of objects but more importantly make both anchors and computed result translation invariant, which means that objects can be recognized, even when their position varies within the images.

The new architecture introduced in Faster R-CNN also impacts how training is done. During training each anchor is assigned a binary class label. More specifically all anchors with an IoU overlap of over 70% with respect to any ground-truth box or

the highest IoU overlap with respect to a single ground-truth box are labeled positive. On the contrary all anchors with an IoU overlap below 30% with respect to all ground-truth boxes are labeled negative. Anchors not labeled by applying these rules do not contribute to the training. Note that a single label of a ground-truth box therefore may be used to assign labels to multiple anchors. Other than that the RPN can be trained end-to-end using stochastic gradient descent and backpropagation. The training described in the paper takes an alternating approach to accomplish that. First the RPN is trained end-to-end for region proposal. Afterwards its output is used to train the detector network. The trained convolutional layers of the detector network are then used to further fine-tune the RPN. At this point the convolutional layers are shared and no longer being trained. Finally while still keeping the convolutional layers fixed, the fully-connected layers of the detector network are fine-tuned.

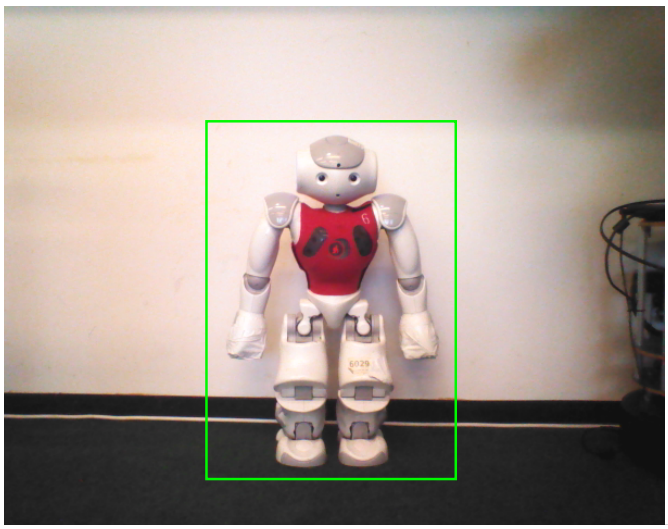
Considering the same conditions as the previous work, Fast R-CNN with RPN was $10\times$ faster at test-time compared to Fast R-CNN with EdgeBox (per image) on the VOC 2007 dataset (see [25]).

4 Robot Detection with Faster R-CNN

Robots used in the standard league of the RoboCup for playing soccer rely on the ability to recognize and locate objects on the field. For instance, a player needs the position of the goal, the ball and the keeper to make a targeted shot against the opponent's goal. Since object detection plays such an important role, teams capable of implementing object detection in their robots with high accuracy and efficiency have a clear competitive advantage. That is why RoboCup participants are constantly trying to improve the accuracy and efficiency with which their players can detect objects during soccer matches. In recent years, methods involving deep learning have shown great results for object detection in terms of accuracy as well as efficiency. This raises the question of whether current deep learning methods for object detection might be a viable choice for object detection in the context of RoboCup. In this section we will evaluate how a current state-of-the-art object detection method based on deep learning performs for detecting robots used in the RoboCup Standard League. We will look at the time needed to train a model with new data and examine the accuracy and efficiency with which this model detects robots in images. In the end, we will compare the method a more traditional method to detect robots.

4.1 Robot Dataset

In order to train and test a new model for robot detection, it is first necessary to gather new data. The image data was provided by the RoboCup team of the Humboldt University of Berlin and prepared in the course of a diploma thesis [16] with a comparable background to this work. In total there are about 1800 images containing robots. The images were taken from different distances, so that the robots can be seen in different sizes. Furthermore, the robots can be seen at different positions and in different quantities in the images. It is worth noting that the provided dataset distinguishes between the perspectives from which the images were taken from. It distinguishes between images taken from an aerial viewpoint and images taken from the camera of the robot. Images taken from an aerial viewpoint primarily focus on playing fields during soccer matches of robots. In these pictures the size of the robots does not vary much, but the number and positions of the robots varies considerably. Images taken from the robot's perspective on the other also vary in size depending the distance from which the robot took the images. The rate with which the robot took the images, however, led to a number of images that have very similar content. This negatively impacts the overall diversity of images that were taken from the robot's perspective. Annotations for each of theses images has also been provided. In order to comply with the restrictions given by the implementation, all existing annotations were converted into a format that is complying with the PASCAL VOC annotation format. In this case, objects have to be annotated as a rectangle with two x-coordinates and two y-coordinates. Figure 7 shows an example of an image and its annotation. In an effort to further extend the dataset, about 1200 images without robots, taken from the original PASCAL VOC dataset 2007, were added as well. The final dataset was



```
<annotation>
  <folder>VOC2007</folder>
  <filename>101059.png</filename>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>640</width>
    <height>480</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>robot</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>224</xmin>
      <ymin>113</ymin>
      <xmax>442</xmax>
      <ymax>440</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 7: An example image and its annotation.

divided into a training set of about 600 images with robots and about 600 without robots and a test set of about 1200 images with robots and 600 without robots.

4.2 Implementation

The experiments are entirely based on a Python reimplementation of the official Faster R-CNN code.[1] There are slight differences between the official MATLAB code published with the paper and the Python reimplementation. Python performs slightly slower at test time, and the models created with this reimplementation may vary in their precision from the models presented in the official paper. It is largely based on the deep learning framework Caffe [14], a well-known and widely used machine-vision library written in C++ with Python and MATLAB interfaces. Caffe also supports CUDA, which enables faster processing on CUDA-enabled graphics processing units.

4.3 Training

Models in this work have been trained using the alternating optimization algorithm from the official Faster R-CNN paper of 2015. In order to estimate how the training length affects the resulting model, models were trained with different quantities of iterations. Note that each iteration represents a single stochastic gradient update to the weights of a model. Besides that, most configuration options were adopted from the default configuration found in the implementation. The initial model used for fine-tuning was the Zeiler and Fergus model [30]. Training was accelerated by using a CUDA-enabled graphics card with 6 gigabytes of memory. Table 1 contains estimates of training times for models trained with different quantities of iterations.

[1]The source code is publicly available at https://github.com/rbgirshick/py-faster-rcnn.

Iterations	10000	20000	40000	120000
Training Time (in minutes)	19	37	71	201

Table 1: An overview of training times for models trained with different quantities of iterations in minutes.

4.4 Evaluation

The evaluation takes place in four steps. The first step shows how the average precision (AP) relates to the number of iterations with which a model was trained. The next step is to examine in more detail how the losses of a particular model have developed during his training. In the third step, we will take a closer look at the precise-recall curve of this model. In the last step we will look at some examples of detection.

We will start the evaluation by analysing how the AP relates to the number of iterations with which a model was trained. The AP in this work is calculated as the area under the precision-recall curve. In order to understand precision and recall it is necessary to introduce new terminology. Object detection results can be categorized into four different categories: true positive, false positive, true negative and false negative. True positives are correctly detected objects, whereas false positives are wrongly detected objects. In this case detected objects are labeled true positive, if their IoU value is above a certain threshold with respect to the ground-truth bounding box. Detected objects below that threshold are labeled false positives. False negatives on the other hand are existing objects that have not been detected. True negatives are all non-existing objects that have also not been detected. The precision measures the ration of correctly detected objects among all detected objects. This gives an idea about how precise the detector is when it has detected an object. The recall on the other hand measures the ration of correctly detected objects among all existing objects. The precision-recall curve represents the relationship between these two measures with respect to different confidence thresholds for object detection. Note once again, we have a two-step process, a region proposal step in which we suggest regions that we can use to calculate IoU values with respect to ground-truth boxes and a classification step to calculate confidence scores. In order to calculate the precision-recall curve of a test set of images, all predicted bounding boxes are first sorted by their confidence score from top to bottom. Afterwards bounding boxes are cumulatively added up as true positive, if their IoU value exceeds 0.5 with respect to the ground-truth box. If their IoU value is below 0.5, they are considered false positive and cumulatively added up as such. This yields two lists containing amounts of true positives and false positives with respect to the confidence score for the predicted bounding boxes.

In the first step we will focus on the area below that curve, the average precision. It is a first measure for the quality of our robot detection based on our test dataset. Figure 8 shows the relationship between AP and numbers of iterations with which models were trained. The AP quickly increases for models trained with between 100 and 5000 iterations up to an AP of about 0.65 for the model trained with 5000 iterations.

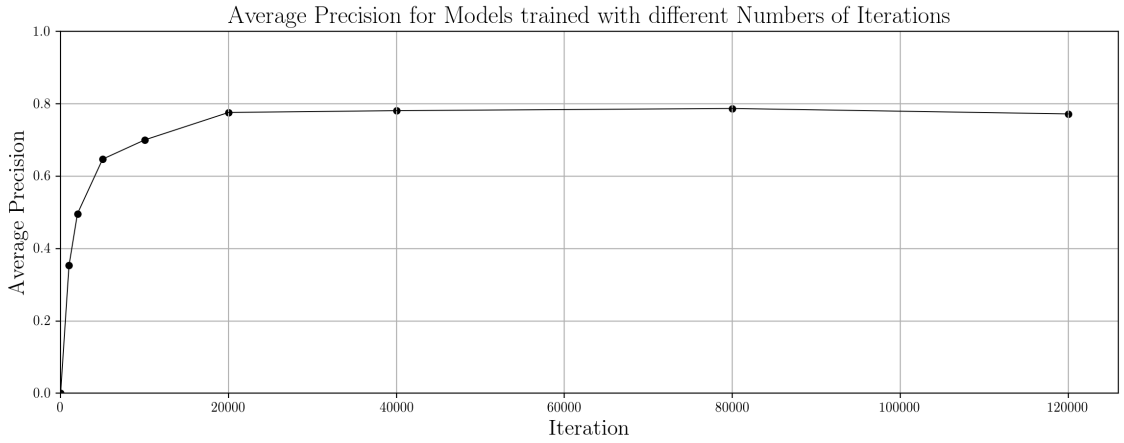


Figure 8: The development of the average precision with respect to the number of iterations with which models were trained.

The AP increases noticeably slower for models trained with between 5000 and 20000 iterations, where the model trained with 20000 achieves an AP of about 0.77 on the test dataset. This implies that the additional 15000 iterations between the model trained with 5000 iterations and the model trained with 20000 iterations only increased the AP by 0.12. Furthermore, we can see that the AP of models trained with a greater number of iterations hardly increases at all. The model trained with 80000 iterations reaches the highest AP among all tested models with about 0.79. Finally, the AP of the last tested model, trained with 120000 iterations even slightly decreased, down to an AP of about 0.77. This suggests that the model possibly cannot be optimized any further after about 80000 iterations using this training dataset. In the next step we

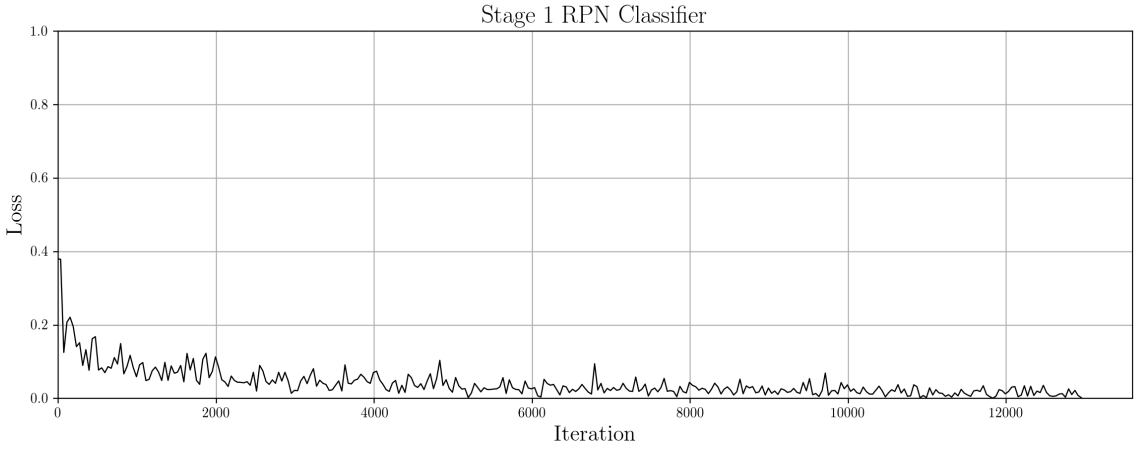


Figure 9: The losses of the RPN's classifier in the first stage of alternating optimization.

will take a look at how the losses of a specific model developed during the course of its training. As described in section 3.3, losses hereby refers to the losses of the classifier and bounding box regressor of the RPN and the classifier and bounding box regressor of the detector network respectively. Since alternating optimization trains the network in separate stages, we will look at the development of the losses in each stage individually. The model chosen for this purpose was trained with 40000 iterations and reaches an AP of about 0.78 on the test dataset. Figure 9 shows the losses of the RPN's classifier in the first stage. The losses quickly decrease til 1000 iterations, where they fluctuate between about 0.15 and 0.5. The losses decline more gradually throughout the rest of the stage and fluctuations diminish to between 0 and 0.03 around 12000 iterations. The losses of the RPN's bounding box regressor as seen in Figure 10 show a similar

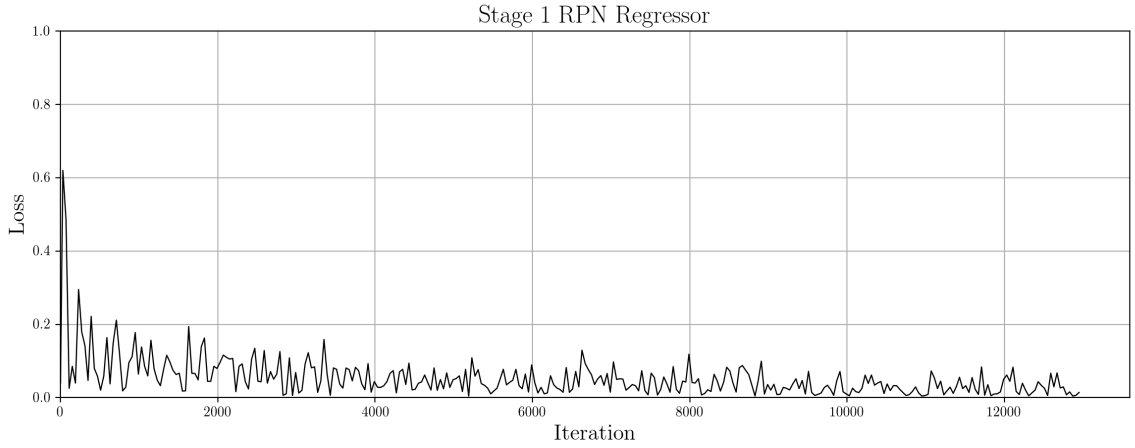


Figure 10: The losses of the RPN's bounding box regressor in the first stage of alternating optimization.

behaviour. The losses start off at about 0.6 and quickly decrease til 1000 iterations, where they fluctuate between about 0.2 and 0. Overall the fluctuations of the losses of the RPN's bounding box regressor seem higher compared to the RPN's classifier. Furthermore, it looks like the fluctuations of the graph do not decrease any further after 2000 iterations but stay about the same between 0 and 0.1. The second stage trains the classifier and the bounding box regressor of the detector network, including the shared convolutional layers of the entire model. We will start by looking at the losses of the detector networks classifier shown in Figure 11. Note that stage 2 and 4 were trained with about half as many iterations as stage 1 and 3. This was adopted from the default configuration of the implementation. The losses of the detector networks classifier start slightly lower with about 0.3 compared to the losses of the RPN's classifier as seen in stage 1. Over the course of stage 2, losses only decrease slowly and fluctuate between about 0 and 0.1 around 6500 iterations. The losses of the detector networks bounding box regressor as shown in Figure 12 slightly decrease till 1000 iterations, where they fluctuate between 0 and 0.3. Afterwards the losses barely decrease and the fluctuations

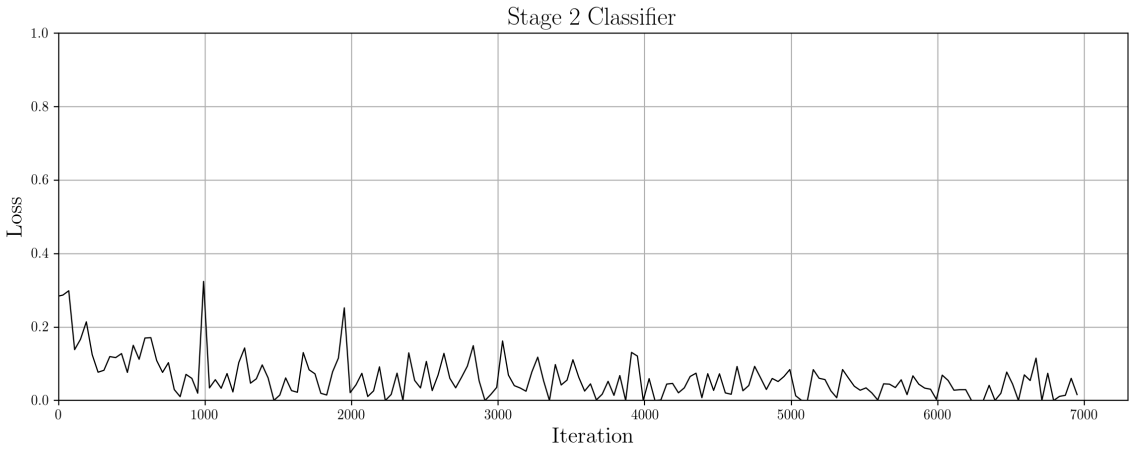


Figure 11: The losses of the detector networks classifier in the second stage of alternating optimization.

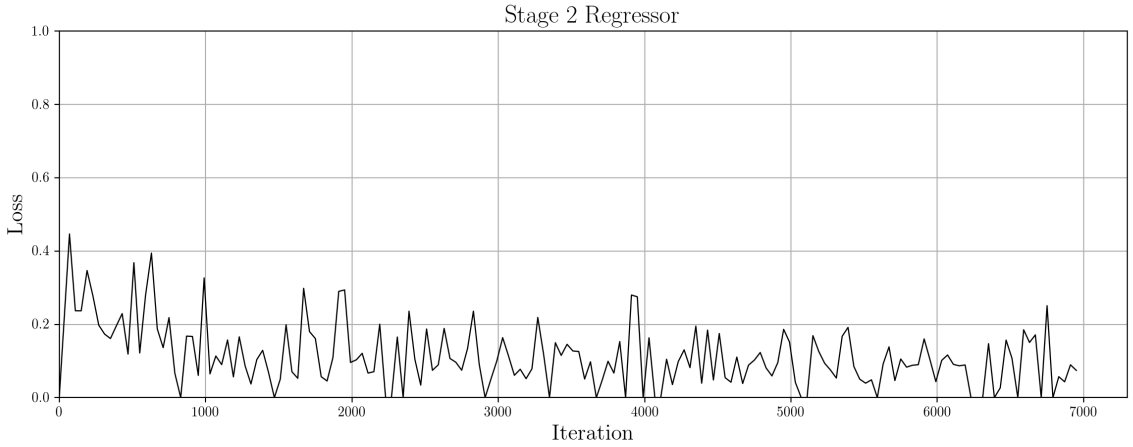


Figure 12: The losses of the detector networks bounding box regressor in the second stage of alternating optimization.

decrease only slightly to between about 0 and 0.2 around 6500 iterations. The third stage again trains the RPN using the trained shared convolutional layers from the previous stage. Note that the shared convolutional layers stay fixed and only the RPN is trained in this stage. We will again start with the losses of the RPN's classifier presented in Figure 13. The losses of the RPN's classifier start of at about 0.3, which is slightly lower than the initial losses in stage 1. Besides that the development is mostly similar to stage 1. However, the fluctuations around 12000 iterations are between about 0 and 0.05, which is higher than the fluctuations around 12000 iterations of stage 1. The losses of the RPN's bounding box regressor shown in Figure 14 start of at about

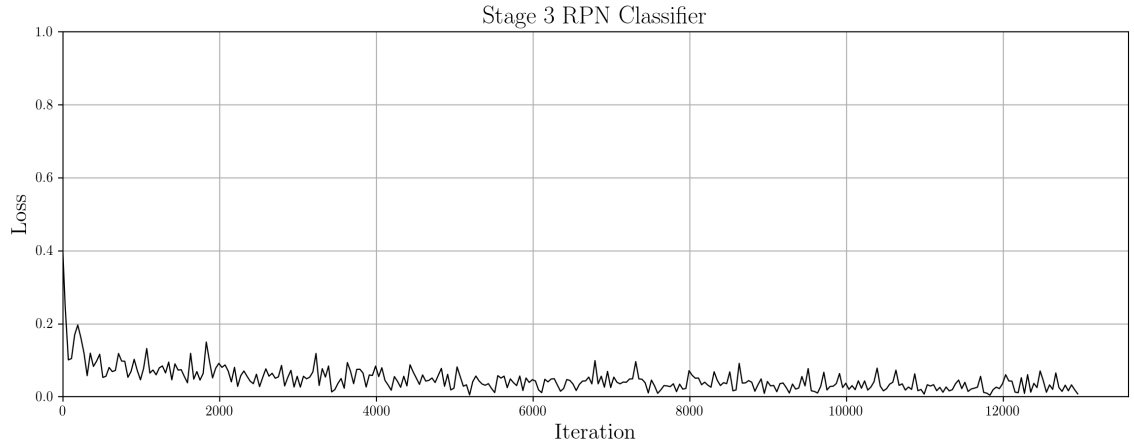


Figure 13: The losses of the RPN's classifier in the third stage of alternating optimization.

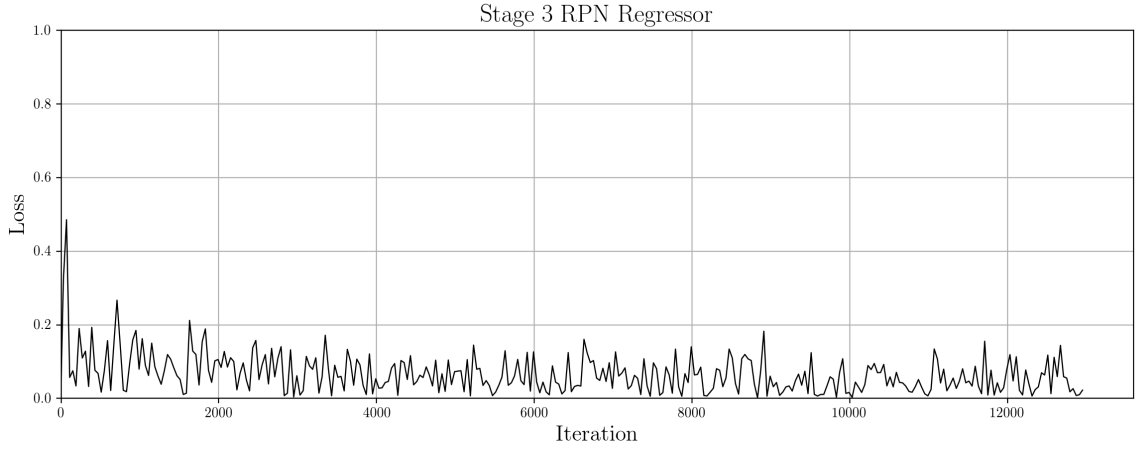


Figure 14: The losses of the RPN's bounding box regressor in the third stage of alternating optimization.

0.5, which is also slightly lower than the initial losses in stage 1. Unlike the losses in stage 1, however, the losses of the bounding box regressor in stage 3 do not seem to decrease throughout the training. They still fluctuate between around 0 and 0.15 around 12000 iterations. The fourth and last stage fine-tunes the detector networks classifier and bounding box regressor using the trained RPN from the previous stage. The losses of the detector networks classifier shown in Figure 15 start at about 0.1, which is lower compared to the initial loss of the second stage. Throughout the entire stage the fluctuations of the losses do not seem to decrease but stay consistent between around 0 and 0.1. The losses of the detector networks bounding box regressor shown in Figure 16 also start lower at about 0.3 compared to the second stage. The losses

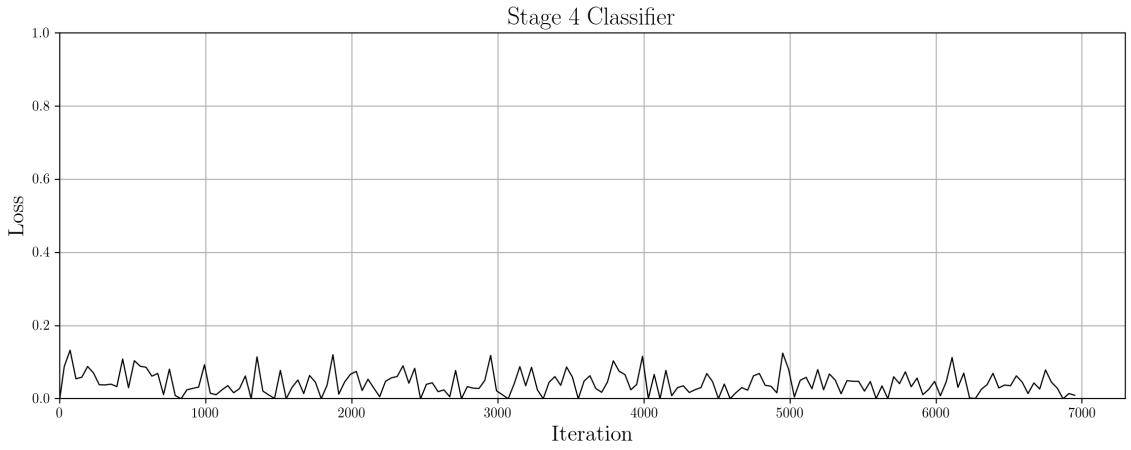


Figure 15: The losses of the detector networks classifier in the last stage of alternating optimization.

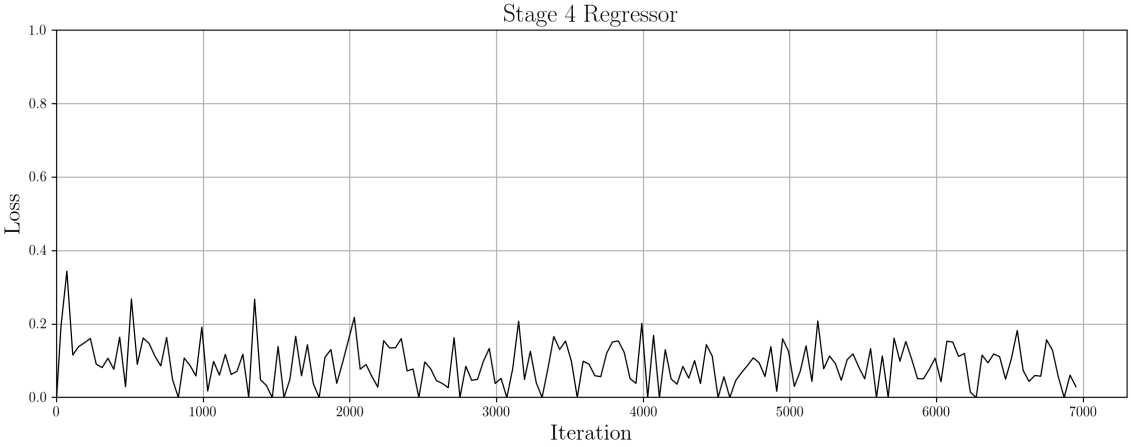


Figure 16: The losses of the detector networks bounding box regressor in the last stage of alternating optimization.

only decrease slowly and the fluctuations of the losses similar to the fluctuations of the second stage.

In the third step of this evaluation we will go into more detail regarding the average precision. As mentioned earlier the average precision represents the relation between precision and recall. In some applications one of these measures might tend to be more important than the other. For example, if we were more interested in finding every object within an image and do not mind detecting a few objects wrong, recall is more important than precision. Figure 17 shows the precision-recall curve for the test dataset. Note that the distribution of value pairs is not clearly visible in this figure.

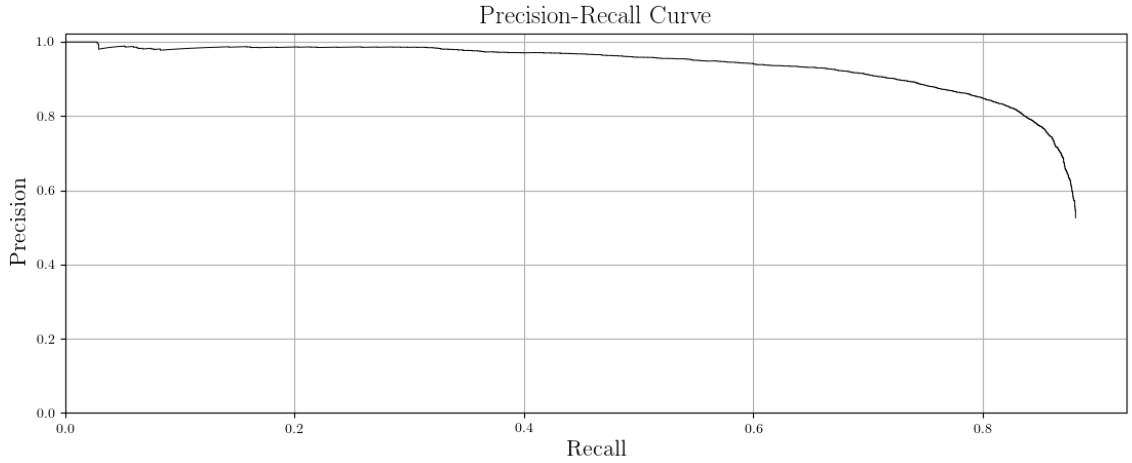


Figure 17: The precision-recall curve for the test dataset.

We can see that the precision only declines slowly. Thus, detections with lower confidence are still mostly correctly detected objects. The precision decreases significantly only for the lowest confidences. Table 2 provides some estimates of precision and recall values for specific confidence thresholds. From this table and the precision-recall curve we can see that the majority of predictions has high confidence scores. Experiments showed that only about 30% of all predictions had a confidence score below 0.5. The maximum tradeoff between precision and recall as measured by their sum is reached for a confidence threshold of about 0.833 with a precision value of about 0.832 and a recall value of about 0.817.

Confidence	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Precision	0.87	0.81	0.77	0.74	0.72	0.68	0.65	0.62	0.57
Recall	0.77	0.83	0.85	0.86	0.86	0.87	0.87	0.87	0.88

Table 2: An overview of precision and recall values for specific confidence thresholds.

In the last step we will take a look at some detection examples. As mentioned earlier the dataset was divided into images taken from an aerial viewpoint and images taken from the robot itself. We will start by looking at detection results for images taken from an aerial viewpoint. Figure 18 shows a playing field with robots playing soccer. Nearly every robot in the image has been detected with high confidence and an accurate bounding box. Even the more difficult to detect robot kneeling behind the goal was detected with confidence above 0.8. The detection results for most images taken from an aerial viewpoint were similarly good during the experiments. However, there are some exceptions. In Figure 19, for example, we can see the influence that lightning has on the detection results. Only two out of all visible robots were detected for the given

Figure 18: Detection results for an image taken from an aerial viewpoint.

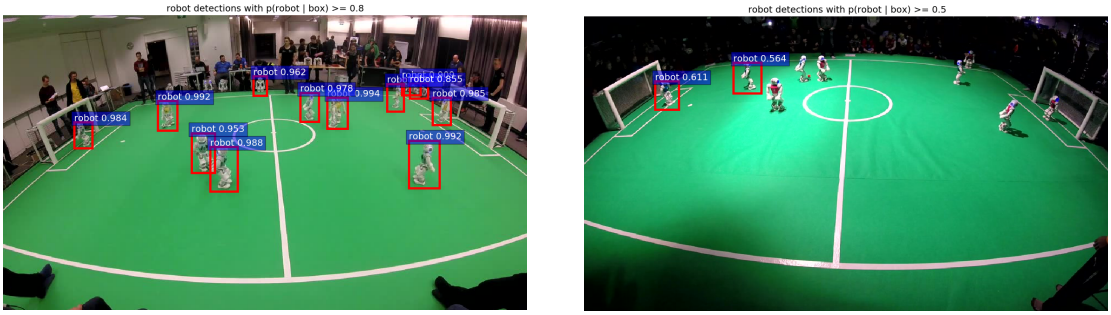


Figure 19: Side by side comparison of images with different lighting conditions.

confidence threshold of 0.5. Furthermore, the two detected robots were only detected with confidences of 0.61 and 0.56. This suggests that the training data possible did not vary enough in terms of lightning conditions. Next we will look at detection results for images taken from the robots perspective. Figure 20 shows three robots in front of a goal as seen by a fourth robot standing in front of them. Similarly as before, many images taken from this perspective show good detection results as seen in Figure 20. However, some images with motives similar to the one seen on the left in Figure 21 repeatedly showed faulty detections. This might relate to the lack of diversity and quantity of training data for images of this kind. On the right side of Figure 21 we can see detection results for an image that was in neither test nor training set. Even though the bounding boxes for the detected objects are mostly good, the confidence is lower compared to detection results for images from the test dataset. Overall, the detection results for images taken from an aerial viewpoint appeared to be better than the results for images taken from the robots perspective.

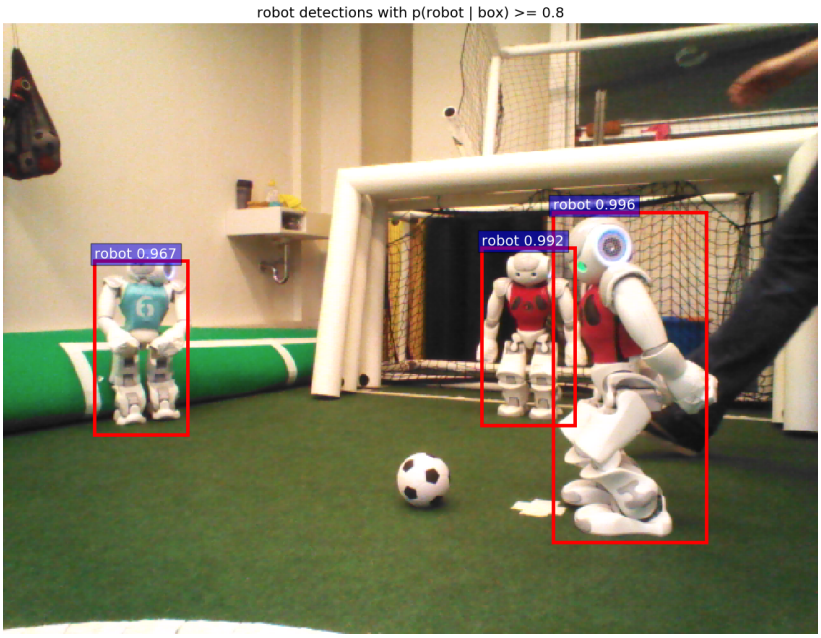


Figure 20: Detection results for an image taken from an robots perspective.

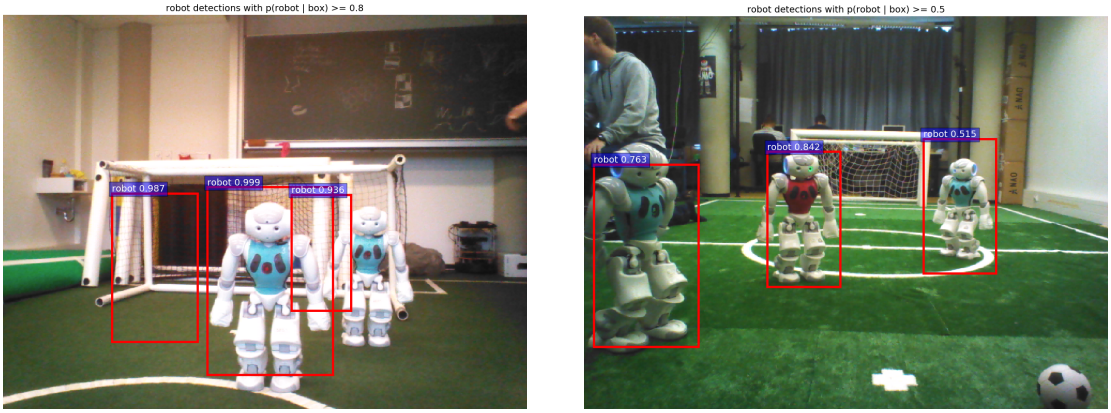


Figure 21: The left side shows faulty detections for an image of the test dataset, while the right side shows detection results for images that were neither in the test dataset nor in training dataset.

5 Conclusion

We started this work by looking at why deep learning is so popular at the moment. After that we took a look at the RoboCup and the extent to which object detection is employed in this particular case. With the objective of testing object detection based on deep learning for RoboCup, we first acquired a basic understanding of machine learning. Here, we primarily focused on a selection of learning algorithms as well as different learning styles. Based on this, we addressed some more specific concepts regarding artificial neural networks and deep learning. In particular, we have gained insights into how artificial neural networks can be optimized and how convolutional neural networks operate. We then extended this knowledge by looking at region-based neural networks. In doing so, we saw how this approach can be used to detect objects in images. Subsequently, we tested the method to detect robots used in the Standard Platform of the RoboCup. Here, we have seen how training data is prepared and how quick a neural network can be trained using specific hardware. In the analysis of the training, we saw how the network improved during the course of the training and how well the different models performed with respect to the number of iterations with which they were trained. Afterwards we took a look at the precision-recall curve and discussed its significance. Last but not least, we looked at the detection results for some specific images.

As we can see, an object detection method that is based on deep learning can successfully be used to detect objects of a new application domain. It became apparent that applying such a method does not require extensive prior knowledge in the field of machine learning or more specifically deep learning. This of course disregards possible improvements to the implementation and data preparation that could have been done, given more knowledge in the field. Furthermore, we can say that applying such a method does not require expensive hardware requirements but can be done using standard hardware. Additionally, we saw that a good model can be trained in little time, which allows users to try out different configurations for this method and find one that fits their application well enough in reasonable time. It is however, worth mentioning that annotating objects in images, especially if on average many objects can be found in the relevant images, can be quite tedious. Since labelling data is required for most if not all currently available object detection methods based on deep learning, this might pose an issue. The closing examples also suggested the influence that quantity and diversity among the training samples has on the final detection results. The results, particularly with regard to new data, would presumably have been better, if there had been more data available for training and more diversity in the data. This may very well apply to similar methods. In this specific case, especially different lighting conditions in the images and the size ratios of the objects in the images may not have been available to sufficient extent in the training data.

An issue that has not been discussed in this work is the implementation of this method on the robots used in the RoboCup. In the RoboCup, no standard hardware such as a GPU might be available. This primarily concerns the detection process and not the training process. Models can be trained on a different machine that has a GPU

and then be transferred onto the robots. Without the usage of a GPU during detection, however, the method needs two seconds per image instead of 200 milliseconds. This increased delay means that the application of this method on the robots may be very limited. It would now be of interest to apply other object detection methods, such as YOLO or SSD, which are supposedly a lot faster than Faster R-CNN, for detection to this task as well to see how they compare.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den November 12, 2017 .....

References

[1] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[2] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[3] J. Constine. NYU "Deep Learning" Professor LeCun Will Head Facebook's New Artificial Intelligence Lab. *TechCrunch. url: https://techcrunch.com/2013/12/09/facebook-artificial-intelligence-lab-lecun/(visited on 06/30/2017)*, 2013.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[5] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[6] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[7] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[8] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[11] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[12] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3): 197–243, 1995.

[13] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[15] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *EMNLP*, volume 3, page 413, 2013.

[16] D. Krienelke. Visuelle detektion humanoider roboter basierend auf histogrammen orientierter gradienten. Diploma thesis., Humboldt-Universität zu Berlin, 2017. To appear.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[18] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[19] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[21] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[22] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[23] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[25] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[26] D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[28] C. Shu. Google acquires artificial intelligence startup deepmind for more than $500M. *TechCrunch. url: https://techcrunch.com/2014/01/26/google-deepmind/(visited on 06/30/2017)*, 2014.

[29] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2): 154–171, 2013.

[30] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[31] C. L. Zitnick and P. Dollár. *Edge Boxes: Locating Object Proposals from Edges*, pages 391–405. Springer International Publishing, Cham, 2014. ISBN 978-3-319-10602-1. doi: 10.1007/978-3-319-10602-1_26. URL `https://doi.org/10.1007/978-3-319-10602-1_26`.