TUHH
Technische Universität Hamburg-Harburg

ICS
Institute of Control Systems

Master's thesis

Object Localization on the Nao Robotic System Using a Deep Convolutional Neural Network and an Image Contrast Based Approach

Author:
Nathapon Olaf Lüders

Supervisor: M. Sc. Patrick Göttsch

Examiner: 1. Prof. Dr. Herbert Werner
2. Prof. Dr.-Ing. Rolf-Rainer Grigat

15. August 2016

ICS
Institute of Control Systems

Master's thesis | **M - 2 - 2016** | Study Course: IMPMEC
for Mr. Nathapon Olaf Lüders | Student ID: 44250

Title:
Object Localization on the Nao Robotic System Using a Deep Convolutional Neural Network and an Image Contrast Based Approach

Project Description:
For every autonomous system it is necessary to gain knowledge of its environment using sensors. The Nao robotic system uses cameras to recognize its environment and extract information from the pictures. On the soccer field with the help of kinematic data the robot gains knowledge of the ball, the goal, field corners, other robots and obstacles. In previous works [1, 2, 3] several vision algorithms were investigated.
A more general approach to the problem of gaining information through vision systems is to solely localize possible regions of interest within images captured by the robot's camera, followed by a classification of the generated regions. This also leads to a more efficient use of the limited computational power on real-time systems.
This thesis focuses on the localization of possible objects without any classification. Two possible and different approaches are investigated: One solution involves making use of contrast values within images to detect saliency regions, which corresponds to the way humans recognize image regions [4, 5]. An alternative solution for this problem involves a more advanced technique: an artificial neural network (ANN). ANNs do not necessarily need specific input values in the form of image features like e. g. contrast or edges but take the raw image and learn how to interpret the image data. Therefore no preceding step of feature generation and selection is necessary. Deep convolutional neural networks (DCNN) recently achieved state-of-the-art performance in the field of object detection [6, 7, 8].

Tasks:
1. Literature survey about ANN and other approaches in the field of object localization
2. Implementation of a DCNN for object localization on a robot soccer field using camera images of the Nao
3. Generate training data and train the DCNN
4. Implementation of an alternative algorithm, see [4, 5]
5. Optional: Test both algorithms on the Nao robotic system
6. Comparison of the results
7. Evaluation and discussion

References:

[1] Erkennung und Klassifizierung von Objekten zur Selbstlokalisierung durch digitale Bildverarbeitung unter Verwendung der integrierten HD-Kameras des NAO-Robotiksystems, Nico Holst, 20.08.2012

[2] Erkennung und Lokalisierung nicht statischer Objekte durch digitale Bildverarbeitung unter Verwendung der integrierten HD-Kameras des NAO-Robotiksystems, Oliver Tretau, 15.12.2012

[3] Implementierung und Verfikation einer Selbstlokalisierung unter Verwendung der integrierten HD-Kameras des humanoide NAO-Robotiksystems, Andre Wurl, 16.12.2012

[4] Global Contrast based Salient Region Detection, Ming-Ming Cheng, Niloy J. Mitra, Xiaolei Huang, Philip H. S. Torr, and Shi-Min Hu, 2014

[5] BING: Binarized Normed Gradients for Objectness Estimation at 300fps Ming-Ming Cheng, Ziming Zhang, Wen-Yan Lin, and Philip Torr, 2014

[6] Scalable Object Detection using Deep Neural Networks, Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov, 2014

[7] You Only Look Once: Unified, Real-Time Object Detection, Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

[8] SSD: Single Shot MultiBox Detector, Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed

Supervisor: M. Sc. Patrick Göttsch

Examiner: 1. Prof. Dr. Herbert Werner
2. Prof. Dr.-Ing. Rolf-Rainer Grigat

Start Date: 16.02.2016
Due Date: 15.08.2016

15. August 2016, Prof. Dr. H. Werner

Hereby I declare that I produced the present work myself only with the help of the indicated aids and sources.

Hamburg, 15. August 2016 Nathapon Olaf Lüders

Abstract

This work compares the performance of two machine learning methods for object localization on images. For this purpose a support vector machine and a convolutional neural network are applied to the problem of localizing Nao robots on a robot soccer field. To gain more insight into the learning behavior of the neural network not only the final performance but also the internal processing is explored. Furthermore data augmentation techniques are used and evaluated in their effect.

Table of Contents

List of Figures

List of Tables

List of Acronyms

DCNN Deep Convolutional Neural Network

ANN Artificial Neural Network

SPL Standard Platform League

SVM Support Vector Machine

ReLU Rectified Linear Unit

RoI Region of Interest

IoU Intersection over Union

TP True Positive

TN True Negative

FP False Positive

FN False Negative

NG Normed Gradient

BING Binarized Normed Gradient

NMS Non-Maximum Suppression

FLOP Floating Point Operation

FLOPS Floating Point Operations Per Second

FPS Frames Per Second

HSV Hue, Saturation, Value

RGB Red, Green, Blue

Introduction

Machine learning gains more and more importance in current digital data processing. This can be traced back to the increase of accessible computational power and data storage of modern computer systems. The Internet is a further reason for the success of such methods. It gives a wide range of people and facilities access to large sets of data, which in turn can be processed by powerful computers and cloud computing. It also enables institutions to collect data, which need human input or expertise e.g. *Amazon Mechanical Turk* or *CAPTCHA*-Codes.

The distinctive feature of machine learning is the ability of such methods to learn relevant patterns of data sets instead of filtering through humanly designed rules. This makes machine learning on the one hand a very powerful tool but on the other hand less predictable and thereby less reliable in safety critical environments. To utilize the full potential of these methods it is important to gain insight into the learning process so that the reason for unexpected outputs can be amended before they arise. A large field of machine learning research is related to image processing. There are also research competitions for image classification and object detection as well as segmentation [1], [2]. A recently very well performing but less intensively researched and understood method in those competitions is the deep convolutional neural network (DCNN) [3], [4]. It is based on the biologically inspired idea of an artificial neural network (ANN). ANNs consist of simplified mathematical models of biological neurons, which mimic the behavior of their biological counterparts. Then a network is created by cross-linking multiple instances resulting in a system, which is able to be trained to perform complex decision. DCNNs have the same basic behavior but are closer to the visual cortex of animals because they look for small certain features on images in overlapping regions [5].

The aim of this work is to utilize machine learning to extract potentially interesting regions of images, so that these small image parts can be processed and evaluated with more specific and powerful methods. The scenario for this research is robot soccer of the RoboCup Standard Platform League (SPL) [6]. Because this is a real time application with limited computational power, it is necessary to utilize fast and small dimensioned approaches. This offers in turn more insight into the performance and potential of machine learning methods and especially DCNNs, when only small networks can be applied to a problem. Therefore two different machine learning methods are tested and eventually compared in their performance - a DCNN and an approach using support vector machines (SVM) on contrast images. Although the applied approaches are intended to find generalized objects, the focus lies on localizing robots, so that a more detailed perspective of the learning process can be given.

The following chapter 1 explains the fundamental principles behind both methods and introduces measures to evaluate the performance. Chapter 2 presents the general constraints for both approaches and describes the reasoning behind their selection as well as their implementation. Chapter 3 gives and overview over typical issues of training in machine learning and discusses solutions. Furthermore, the image data base and training methodology of both approaches is presented. Both approaches are then evaluated with respect

to the posed problem and the learning characteristics in chapter 4. Lastly, a concluding summary of this work and an outlook for further research is given in chapter 5.

1 Fundamentals

This chapter explains the fundamental principles of this work. This includes the target hardware, methods and evaluation units.

1.1 Nao Robotic System

The Nao is a programmable humanoid robot produced and sold by *Softbank Robotics*. It is 0.57 m tall and weighs 5.2 kg [7]. It is composed of many various sensors including two cameras, speakers, microphones, haptic sensors, a gyroscope, infra red and sonar. Since this work is only focused on image processing, only camera relevant topics will be discussed.

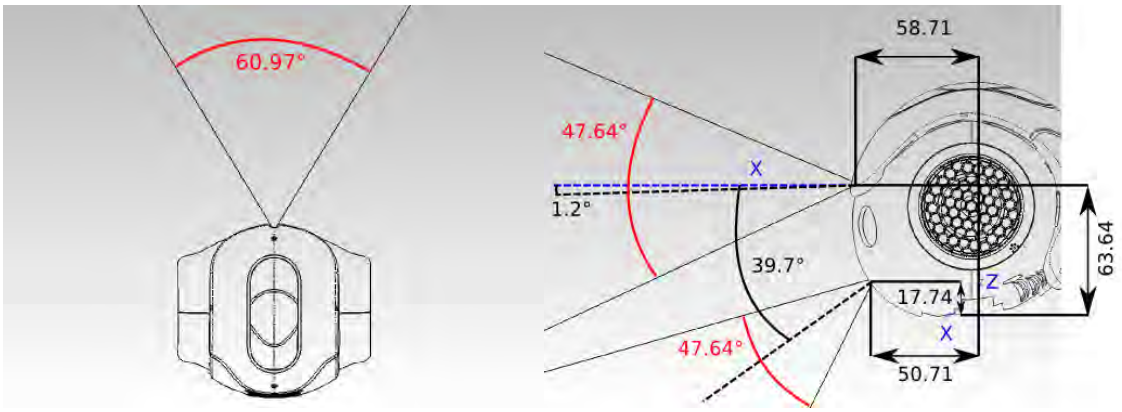


Fig. 1.1: Nao head with dimensions and properties of its two cameras [7].

Similar to humans the head contains the vision system and the information processing unit. The vision system is composed of two cameras - one in the forehead and one at the position of the mouth (s. Figure 1.1). The field of vision of both cameras only overlaps partially, thus they do not produce a full stereo vision image. The information processing unit is an Intel ATOM Z530 1.6 GHz CPU with 1 GB RAM.

1.2 Machine Learning

Machine learning describes the process of knowledge generation from experience. I.e. that a system learns generalized patterns and rules from examples, which can then be applied to unknown data. This has the benefit of learning dependencies between input signals, which might not be obvious to a human. Therefore it can be useful to apply machine learning techniques to problems with very little or no prior knowledge of input signal dependencies. Additionally machine learning and especially supervised learning with available ground truth data for each training example requires minimal human interaction and can be performed on a large scale using modern computers.

In this work, two machine learning approaches are applied to the problem of object localization, whose fundamental principles are explained in the following sections.

1.2.1 Support Vector Machine

The following and the subsequent sections about SVMs are based on [8], if not otherwise mentioned. SVMs are basically a method to classify linearly. They separate two classes in a d-dimensional space through a separating hyperplane, e.g. a line in the two-dimensional case (s. Figure 1.2 below).

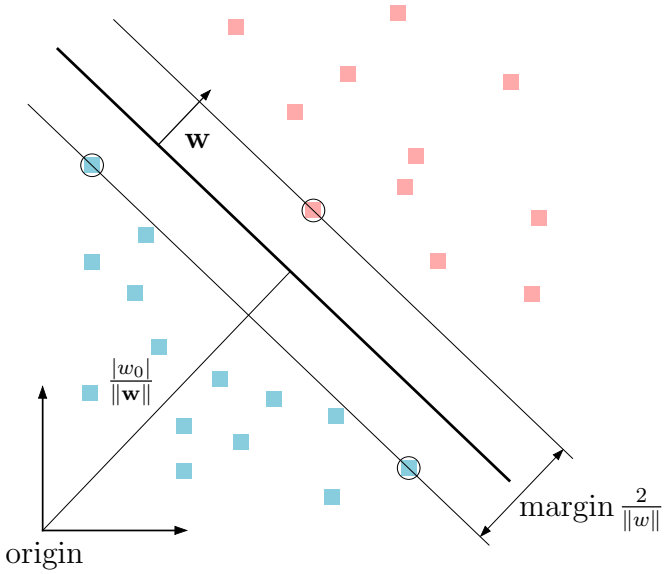


Fig. 1.2: Optimal separating hyperplane in a two-dimensional case. The encircled instances are the defining support vectors.

Hyperplanes cannot be warped which is why the classes have to be linearly separable. The hyperplane is only defined by the vectors of those class instances, which lie closest to the hyperplane. These vectors are called support vectors and justify the name of this method. The distance between hyperplane and support vectors is maximized by optimization methods for optimal separation.

Assuming two separate classes C_1 and C_2 with corresponding labels $+1$ and -1, an instance can be described by $\mathcal{X} = \{\mathbf{x}^t, r^t\}$. $r^t = +1$, if $\mathbf{x}^t$ belongs to C_1 ($\mathbf{x}^t \in C_1$) and $r^t = -1$, if $\mathbf{x}^t$ belongs to C_2 ($\mathbf{x}^t \in C_2$) [8]. Thus a $\mathbf{w}$ and w_0 of a hyperplane are sought, subject to the following constraints.

$$\mathbf{w}^T\mathbf{x}^t + w_0 \geq +1 \text{ für } r^t = +1 \tag{1.1}$$

$$\mathbf{w}^T\mathbf{x}^t + w_0 \geq -1 \text{ für } r^t = -1 \tag{1.2}$$

Both constraints can be combined:

$$r^t(\mathbf{w}^T\mathbf{x}^t + w_0) \geq +1 \text{ für } r^t = +1 \tag{1.3}$$

The value +1 on the right side of the inequality describes the minimum distance between hyperplane and the nearest class instance. This increases the generalization ability of the SVM. The distance between hyperplane and an arbitrary class instance $\mathbf{x}^t$ can be calculated by the following expression.

$$\frac{|\mathbf{w}^T\mathbf{x}^t + w_0|}{\|\mathbf{w}\|} \tag{1.4}$$

1.4 is maximized, when $\|w\|$ is minimized. An SVM can be computed by the following optimization rule.

$$\min \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } r^t(\mathbf{w}^T\mathbf{x}^t + w_0) \geq +1, \forall t \tag{1.5}$$

The closest class instances have distance $\frac{1}{\|\mathbf{w}\|}$ to the hyperplane. 1.5 is a quadratic optimization problem, whose complexity depends on the input dimension d and not on the number of training examples N. This results from the fact that the separating hyperplane $\mathbf{w}$ has the dimension of the input space d and is the subject to be optimized with the above formulation. However, the higher dimension causes a more complex optimization problem. To remedy this, 1.5 can be rewritten to an unconstrained problem using Lagrange multipliers α^t.

$$\begin{aligned} L_p &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{t=1}^{N} \alpha^t [r^t(\mathbf{w}^T\mathbf{x}^t + w_0) - 1] \\ &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{t=1}^{N} \alpha^t r^t(\mathbf{w}^T\mathbf{x}^t + w_0) + \sum_{t=1}^{N} \alpha^t \end{aligned} \tag{1.6}$$

L_p must be maximized with respect to α^t, subject to the constraints that the gradient of L_p with respect to $\mathbf{w}$ and w_0 are zero and that $\alpha^t \geq 0$. Because this is a convex quadratic optimization problem, the *Karush-Kuhn-Tucker* constraints of the following two equations can be applied to solve the dual problem.

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{t=1}^{N} \alpha^t r^t \mathbf{x}^t \tag{1.7}$$

$$\frac{\partial L_p}{\partial w_0} = 0 \Rightarrow \sum_{t=1}^{N} \alpha^t r^t = 0 \tag{1.8}$$

Plugging 1.7 and 1.8 into equation 1.6 yields a dual problem L_d with different constraints:

$$\begin{aligned} L_d &= \frac{1}{2}\left(\mathbf{w}^T\mathbf{w}\right) - \mathbf{w}^T \sum_{t=1}^{N} \alpha^t r^t \mathbf{x}^t - w_0 \sum_{t=1}^{N} \alpha^t r^t + \sum_{t=1}^{N} \alpha^t \\ &= -\frac{1}{2}\left(\mathbf{w}^T\mathbf{w}\right) + \sum_{t=1}^{N} \alpha^t \\ &= -\frac{1}{2} \sum_{t=1}^{N} \alpha^t r^t \left(\mathbf{x}^t\right)^T \sum_{t=1}^{N} \alpha^t r^t \mathbf{x}^t + \sum_{t=1}^{N} \alpha^t \end{aligned} \tag{1.9}$$

1.9 must be maximized with respect to α^t and subject to $\sum_t \alpha^t r^t = 0$ und $\alpha^t \geq 0, \forall t$. The problem can then be solved with quadratic optimization methods.

If α^t is calculated for all N, the major part vanishes with $\alpha^t = 0$. Only a small part of $\mathbf{x}^t$ with $\alpha^t > 0$ remains and defines the set of $\mathbf{x}^t$, which are stored as support vectors of the SVM. The support vectors fulfill the constraint $r^t(\mathbf{w}^T\mathbf{x}^t + w_0) = 1$ and have the minimum distance to to the separating hyperplane. From this constraint w_0 can be computed using any support vector.

$$w_0 = r^t - \mathbf{w}^T\mathbf{x}^t \tag{1.10}$$

When performing classification, the minimum distance is not enforced anymore, so that the classification rule becomes:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{1.11}$$

$$\mathbf{x} \in C_1 \text{ if } g(\mathbf{x}) > 0$$
$$\mathbf{x} \in C_2 \text{ if } g(\mathbf{x}) \leq 0$$

Soft Margin Hyperplane

Two datasets are often not linearly separable in real applications. The previously described algorithm will not work in this case because the minimum distance to the separating hyperplane cannot be established. Therefore $\xi^t \geq 0$ is introduced. ξ^t describes the deviation to the minimum required distance. If $0 < \xi^t < 1$, ξ^t is correctly classified but does not have the required minimum distance. If $\xi^t \geq 1$, ξ^t lies on the wrong side of the hyperplane, which results in a wrong classification. If $\xi^t = 0$, ξ^t is correctly classified and has the required minimum distance. The soft margin hyperplane has to fulfill the following requirement:

$$r^t(\mathbf{w}^T\mathbf{x}^t + w_0) \geq 1 - \xi^t \tag{1.12}$$

With C as adjustable parameter to weight the influence of $\sum_t \xi^t$, 1.6 can be reformulated.

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{t=1}^{N}\xi^t - \sum_{t=1}^{N}\alpha^t[r^t(\mathbf{w}^T\mathbf{x}^t + w_0) - 1 + \xi^t] - \sum_{t=1}^{N}\mu^t\xi^t \tag{1.13}$$

μ^t is introduced as new Lagrange parameter to ensure a positive ξ^t. Setting $\frac{\partial L_p}{\partial \mathbf{w}}$, $\frac{\partial L_p}{\partial w_0}$ and $\frac{\partial L_p}{\partial \xi^t}$ to zero and plugging the results into 1.13, yields again the dual problem L_d.

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{t=1}^{N}\alpha^t r^t \mathbf{x}^t = 0 \Rightarrow \mathbf{w} = \sum_{t=1}^{N}\alpha^t r^t \mathbf{x}^t \tag{1.14}$$

$$\frac{\partial L_p}{\partial w_0} = \sum_{t=1}^{N}\alpha^t r^t = 0 \tag{1.15}$$

$$\frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0 \tag{1.16}$$

$$L_d = -\frac{1}{2}\sum_{t=1}^{N} \alpha^t r^t \left(\mathbf{x}^t\right)^T \sum_{t=1}^{N} \alpha^t r^t \mathbf{x}^t + \sum_{t=1}^{N} \alpha^t \tag{1.17}$$

L_d is maximized with respect to α^t, subject to $\sum_t \alpha^t r^t = 0$ and $0 \leq \alpha^t \leq C, \forall t$.

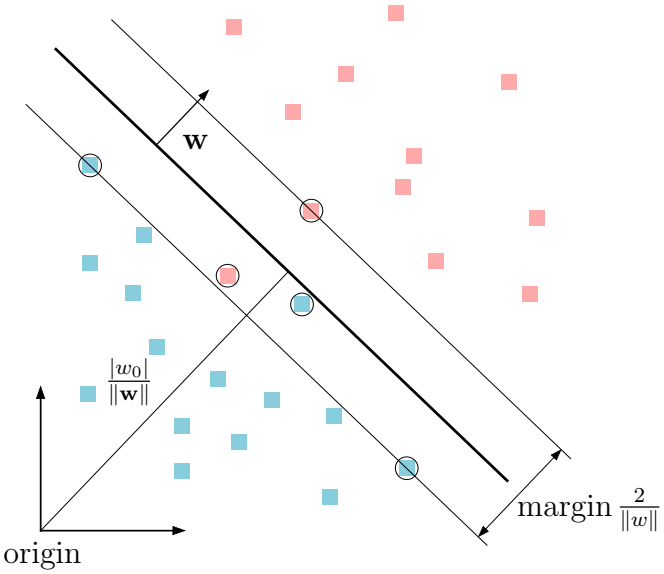


Fig. 1.3: SVM soft margin in a non-linearly separable case. One red instance is on the wrong side of the hyperplane and is misclassified $\xi > 1$. One blue instance lies on the correct side but within the margin $0 < \xi < 1$. The encircled instances are again the support vectors.

Instances with $\alpha^t = 0$ vanish because the distance to the separating hyperplane is sufficient. Instances with $0 < \alpha^t < C$ define $\mathbf{w}$ because they lie exactly on the required margin and are thus the support vectors. Furthermore w_0 can be calculated from $\xi^t = 0$ and $r^t(\mathbf{w}^T\mathbf{x}^t + w_0) = 1$. Instances with $\alpha^t = C$ lie within the margin or are misclassified but also stored as support vectors. Thus the number of support vectors can be interpreted as an upper-bound estimate for the number of errors. 1.11 can again be used to classify instances. Values close to zero would be instances with $\alpha^t = C$ during training and can therefore only be treated with low confidence.

1.2.2 Artificial Neural Networks

The content of this section is based on [8] and [9]. ANNs are networks of highly simplified, artificial neuron models. An artificial neuron is supposed to imitate the behavior of a biological neuron regarding the information processing. Hence the idea is to build a network of artificial neurons to make complex decisions similar to biological life-forms.

For this purpose the model of an artificial neuron is developed from a biological neuron model followed by the description of an ANN.

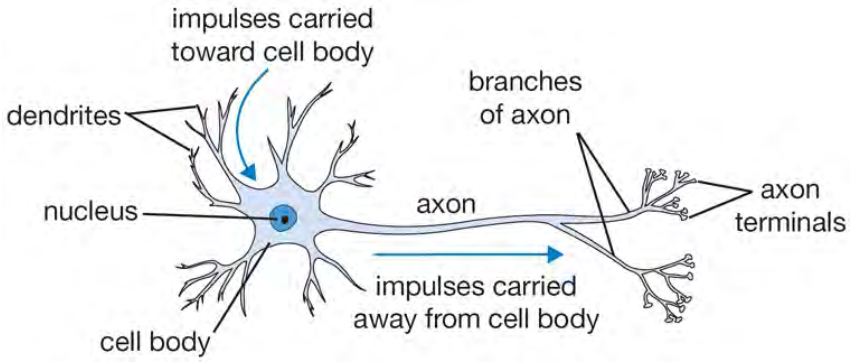


Fig. 1.4: Model sketch of a biological neuron [10].

Figure 1.4 depicts a biological neuron model. Important for the development of an artifical neuron are the dendrites, the nucleus, the axon and the synapses. Dendrites carry electrical signals to the cell body and possess many branches. They can be interpreted as input channels to the neuron. The nucleus processes all input signals and reacts according to its function. If the inputs activate the nucleus, it sends an output signal, which is a sequence of pulses, through its axon. The axon is a single long nerve fiber which is connected to target cells. In the case of an ANN these target cells can be other neurons, which have their dendrites connected to the firing neuron. The connection point between axon and dendrite is called synapse. The connection strength of a synaptic connection may vary depending on the stimulation frequency [11]. A basic model of an artificial neuron can be derived from this description.

Artificial Neuron

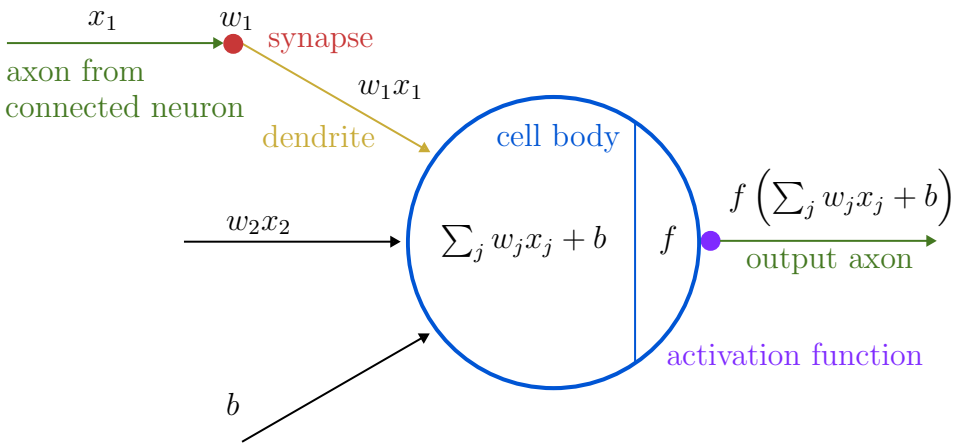


Fig. 1.5: Mathematical model of an artificial neuron based on [10].

Figure 1.5 shows the model of an artificial neuron. w_j is the strength of each synaptic connection, which damps or amplifies the incoming signal x_j of the connected axon. The nucleus processes each weighted signal by building the sum $\sum_j w_j x_j + b$. b is a bias term, which introduces an offset and can shift the threshold for triggering the neuron activation which leads to the next general assumption of this model. The exact timing of each pulse does not have any influence on the activation. Thus the frequency of the output signal is described by an activation function f. Accordingly the axon carries the output signal $f\left(\sum_j w_j x_j + b\right)$. The weights w_j and the bias b are of major importance because they are adjustable parameters and can be trained.

As already mentioned, the bias b can introduce a threshold shift in combination with the activation function f. A typical choice of f is the sigmoid function. An advantage of this function is, that it imitates the behavior of a neuron in the sense that it has an output saturating close to zero, if the input is highly negative, and saturating close to one, if the input has large positive values (s. Figure 1.6).

$$\mathrm{sig}(x) = \frac{e^x}{1 + e^x} \tag{1.18}$$

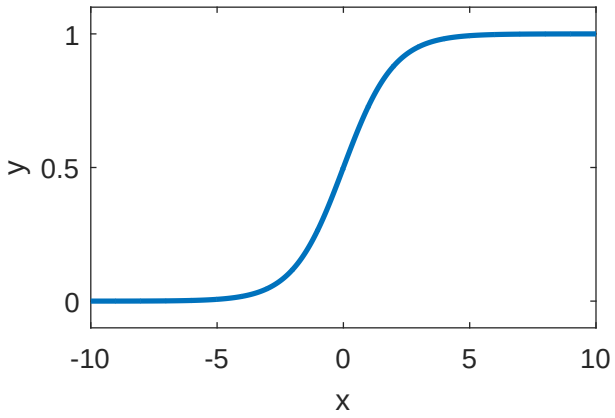


Fig. 1.6: Sigmoid function.

The value of the bias b can shift the activation threshold to the left or right. It also has the mathematically desirable property of differentiability, which is important during training and is discussed later.

$$\frac{\mathrm{d}\,\mathrm{sig}(x)}{\mathrm{d}x} = \mathrm{sig}(x)\,(1 - \mathrm{sig}(x)) \tag{1.19}$$

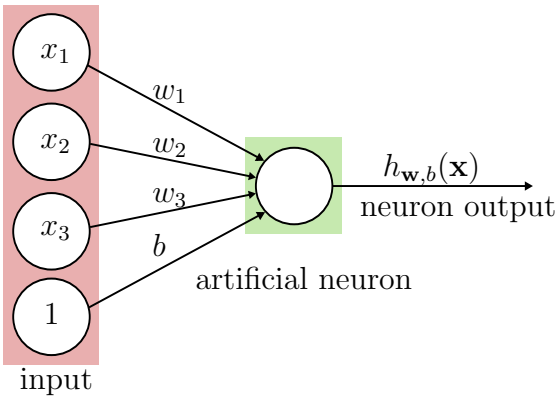


Fig. 1.7: Single artificial neuron with three inputs, a bias term and output $h_{\mathbf{w},b}(\mathbf{x})$. Based on [9].

Figure 1.7 shows a less detailed model of a neuron. This work assumes a real input and parameter space $x, w, b \in \mathbb{R}$ and activation functions f with $f : \mathbb{R} \longmapsto \mathbb{R}$. Given an input the neuron forms a hypothesis $h_{\mathbf{w},b}(\mathbf{x})$ using the weights and bias term:

$$h_{\mathbf{w},b}(\mathbf{x}) = f\left(\sum_{j=1}^{3} w_j x_j + b\right) \tag{1.20}$$

If bias b is assumed to be a weight vector entry w_0 and the constant input 1 an input vector entry $x_0 = 1$, 1.20 can also be written as a more compact vector product with $\mathbf{w} = [b, w_1, ..., w_3]^T$ and $\mathbf{x} = [1, x_1, ..., x_3]^T$.

$$h_{\mathbf{w},b}(\mathbf{x}) = f\left(\mathbf{w}^T\mathbf{x}\right) \tag{1.21}$$

If there are more than one neurons processing the same inputs, they are called parallel. The single inputs and neurons of a layer are also called units. E.g. Figure 1.8 has three input units, three hidden units and one output unit. The units are enumerated from top to bottom starting at one.

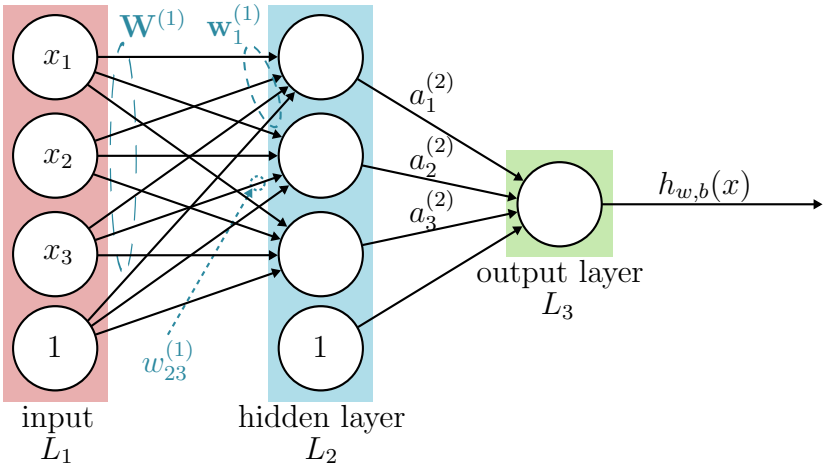


Fig. 1.8: Three layer structure with an input (L_1), hidden (L_2) and output layer (L_3). The input layer as well as the hidden layer have three units (plus one bias term). The blue Ws show how the indices can denote single values, vectors or a matrix of weights. E.g. the index tuple of $w_{23}^{(1)}$ defines the weighted connection between input x_3 and unit 2 of the subsequent layer. $a_1^{(2)}$, $a_2^{(2)}$ and $a_3^{(2)}$ denote the corresponding activations, i.e. neuron outputs. Based on [9].

Let n_l denote the number of network layers and a specific layer l be described by L_l. This makes L_1 the input and L_{n_l} the output layer. As illustrated in Figure 1.8 the weights of a layer can be summarized to weight matrix $\mathbf{W}^{(l)}$. A single weight of a connection between two units can be described by $w_{ij}^{(l)}$, where j denotes the unit of layer l and i the unit of layer $l+1$. If only one lower index is given $w_l^{(1)}$ denotes the weight vector of the connection between all units of layer l and unit 1 of layer $l+1$. The same notation goes for the bias $b_i^{(l)}$ without index j because it is mentioned separately and always connected to a unit outputting 1. The network described above has $n_l = 3$ layers and its parameters can be summarized to $(W, b) = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)})$. Following this notation $a_i^{(l)}$ denotes the activation of unit i in layer l.

The network out of 1.8 can be calculated by the following formulas:

$$\begin{aligned} a_1^{(2)} &= f\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}\right) \\ a_2^{(2)} &= f\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}\right) \\ a_3^{(2)} &= f\left(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}\right) \\ h_{W,b} = a_1^{(3)} &= f\left(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)}\right) \end{aligned} \tag{1.22}$$

The weighted sum of inputs to unit i in layer l, including the bias, can be summarized to $z_i^{(l)}$. Additionally the simplification $a_i^{(1)} = x_i$ can be made to achieve a more generic way of writing down unit activations.

$$a_i^{(l)} = f(z_i^{(l)}) \tag{1.23}$$

Organizing the parameters and inputs in matrices and vectors leads to a very compact and generic formulation for the forward pass of a feedforward ANN of fully connected layers.

$$\mathbf{z}^{(l+1)} = \mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)} \tag{1.24}$$

$$\mathbf{a}^{(l+1)} = f(\mathbf{z}^{(l+1)}) \tag{1.25}$$

The same principles apply for a network with more hidden layers and multiple outputs, making $\mathbf{h}_{W,b}(x)$ a hypothesis vector.

Training

Independent of the size or number of layers of a feedforward ANN it can be trained using backpropagation, which is discussed in the further development of this section. The basic idea is to adjust the weights of the network to minimize a measurable or computable error in the training data. A typical measure is the sum of squared errors. The error for a single example with input $\mathbf{x}$ and ground truth output $\mathbf{y}$ can be calculated by the following equation.

$$J(W, b; x, y) = \frac{1}{2}\|\mathbf{y} - \mathbf{h}_{W,b}(\mathbf{x})\|^2 \tag{1.26}$$

Applying gradient descent to update each network parameter $w_{ij}^{(l)}$ and $b_i^{(l)}$ yields:

$$\begin{aligned} w_{ij}^{(l)} &= w_{ij}^{(l)} - \alpha\frac{\partial J(W, b; \mathbf{x}, \mathbf{y})}{\partial w_{ij}^{(l)}} \\ b_i^{(l)} &= b_i^{(l)} - \alpha\frac{\partial J(W, b; \mathbf{x}, \mathbf{y})}{\partial b_i^{(l)}} \end{aligned} \tag{1.27}$$

α is an adjustable parameter also known as learning rate, which scales the weight update and will be discussed later. To perform gradient descent the partial derivatives of 1.27 have to be computed. This can be achieved with the backpropagation algorithm. Let therefore $\delta_i^{(l)}$ denote an error term that measures how much error was introduced by node i in layer l into the layer output. At first a forward pass is performed using input $\mathbf{x}$ to compute the network hypothesis $\mathbf{h}_{W,b}(\mathbf{x})$. Since ground truth network output $\mathbf{y}$ is known for a training example $\delta_i^{(n_l)}$ can be computed.

$$\delta_i^{(n_l)} = \frac{\partial J(W, b; \mathbf{x}, \mathbf{y})}{\partial z_i^{(n_l)}} = -(y_i - a_i^{(n_l)})f'(z_i^{(n_l)}) \tag{1.28}$$

Once $\delta_i^{(n_l)}$ for the final layer is known, δ_i of the preceding layers can then be computed in backwards propagating fashion, where s_l denotes the number of units in layer l.

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} w_{ij}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{1.29}$$

The necessary partial derivatives for gradient descent weight update of 1.27 are:

$$\begin{aligned} \frac{\partial J(W,b;\mathbf{x},\mathbf{y})}{\partial w_{ij}^{(l)}} &= a_j^{(l)} \delta_i^{(l+1)} \\ \frac{\partial J(W,b;\mathbf{x},\mathbf{y})}{\partial b_i^{(l)}} &= \delta_i^{(l+1)} \end{aligned} \tag{1.30}$$

Vectorization leads again to a more compact notation for the partial derivatives 1.30.

$$\begin{aligned} \nabla_{\mathbf{W}^{(l)}} J(W,b;\mathbf{x},\mathbf{y}) &= \delta^{(l+1)} (a^{(l)})^T \\ \nabla_{\mathbf{b}^{(l)}} J(W,b;\mathbf{x},\mathbf{y}) &= \delta^{(l+1)} \end{aligned} \tag{1.31}$$

Backpropagation and weight updates can be applied in repeated fashion until the error converges to a minimum. It is important to note that the minimum is not necessarily a global minimum. Thus the choice of the learning parameter α is crucial. If α is very small, the weight updates are proportionally small and lead to a slow convergence. A huge α leads to strong weight updates and might never find a minimum because it "leaps" over any local minimum. Additionally the results depend on initialization. A constant initialization might always lead to the same local minimum and similar weight updates. It is recommendable to initialize with small random weights for symmetry breaking and compare the performance of several solutions.

1.3 Artificial Neural Networks in Image Processing

This section describes basic layer types and activation functions commonly used in image processing and are parts of ANNs, which achieved state-of-the-art results [4].

1.3.1 Fully Connected Layer

The details of a fully connected layer were already discussed in section 1.2.2. In DCNNs for image processing they are often used as final or series of final layers processing all extracted features and forming a final decision in a classification or regression problem.

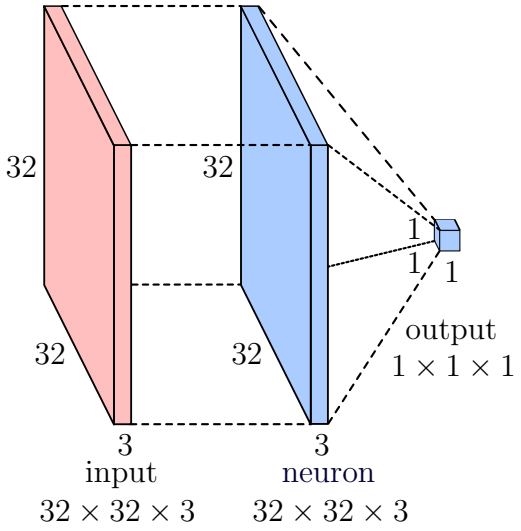


Fig. 1.9: Schematic drawing of a fully connected layer with one neuron (blue) processing an equally sized input (red) and forming a single layer output. The exemplary input dimension of $32 \times 32 \times 3$ could be e.g. an input image with a width and height of 32 pixels and three color channels.

A fully connected layer has as many parameters as its input since it is, as the name states, fully connected to its input plus an additional bias term resulting in $32 \cdot 32 \cdot 3 + 1 = 3073$ parameters (s. Figure 1.9). Thus it introduces a high amount of parameters, if multiple outputs are required or the input dimension is big (s. Figure 1.10 below).

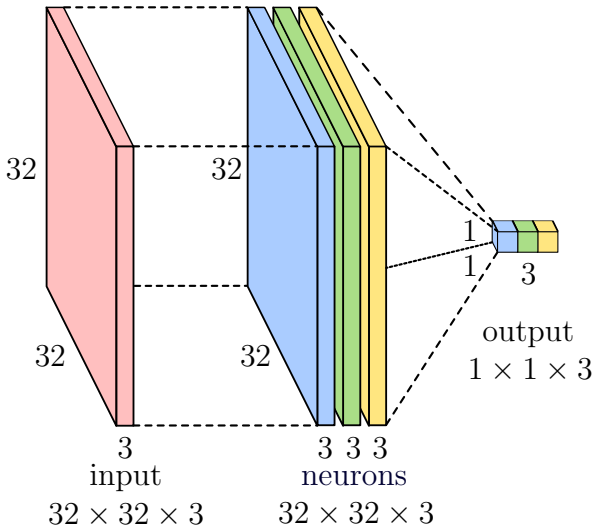


Fig. 1.10: Schematic drawing of a fully connected layer with three neurons (blue, green and yellow) processing the same equally sized input (red) and forming three output values.

Figure 1.10 shows a fully connected layer with three neurons processing the same input. Each neuron (blue, green and yellow) is fully connected to the input (red) and produces

one respective output value independently of the other neurons. The number of trainable parameters rises therefore by a factor of 3.

1.3.2 Convolutional Layer

The convolutional layers are inspired by the organization of the animal visual cortex. Specific cells within the cortex react to edge-like features and are locally invariant to the exact position of patterns [5]. This property is adapted by convolutional neurons in a way that weight parameters are shared making features on images locally invariant which makes these type of layers very feasible. Convolutional neurons "look" for previously learned features e.g. edges or color patches on images and produce a feature map depicting the presence or vacancy of these features. Neurons of convolutional layers have much fewer parameters and can therefore be trained more efficiently. Compared to neurons of fully connected layers, the number of parameters does not depend on the input size but the filter size. As shown below, a neuron of a convolutional layer acts as small filter moving over the input, which results in a feature map instead of a single output. An exemplary filter can have the size of $5 \times 5 \times 3$ and therefore only have $5 \cdot 5 \cot 3 + 1 = 76$ parameters independent of the input width and height, whereas the neuron of a fully connected layer would have $32 \cdot 32 \cdot 3 + 1 = 3073$ parameters in the case of the example input.

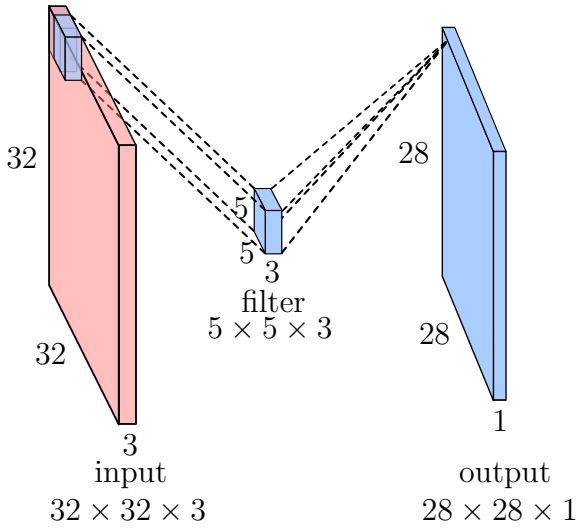


Fig. 1.11: Convolutional Layer performing feature extraction on a $32 \times 32 \times 3$ input.

Figure 1.11 illustrates how the convolution works. On the left side is an input, which could be an image of dimension $32 \times 32 \times 3$. In the middle is a filter also called kernel, which corresponds to the receptive field of a cell. This filter computes the dot product between the small part it is "looking" at and its weight values. Likewise to convolution it goes over the entire input, computing a feature map, which is on the right side in blue. The filter dimensions can be arbitrarily chosen as long as one dimension is not larger than the respective image dimension. The stride with which the filter moves can also be freely chosen but should be smaller than the filter size because otherwise the

desired property of overlapping receptive fields vanishes. Depending on the filter size and stride the convolution not only reduces the depth but also width and height dimension (e.g. from $32 \times 32 \times 3$ to $28 \times 28 \times 1$ in example figure above). Large network design is often easier, if the output width and height is equal or an integer fraction of the input. To deal with this problem modern libraries apply *Zero Padding*. *Zero Padding* frames the input with zeros, such that the output has the desired size after convolution. Since a dot product is computed the zeros do not influence the resulting feature map but introduce additional mathematical operations, which are not feasible on systems with tough hardware constraints. Thus *Zero Padding* is not used or further discussed in this work. Likewise to fully connected layer, convolutional layers can also have multiple neurons in form of filters processing the same input. This is also very common because one filter learns to extract the presence of one feature e.g. a vertical edge on an input image. Another filter would then be necessary to detect e.g. horizontal edges or a certain color. Each filter produces its own feature map depicting the presence of the respective feature it is trained to detect. A subsequent convolutional layer can process these feature maps and detect deeper features, which are combinations of the previously detected feature e.g. a vertical edge with a specific color. These deep features are responsible for the name of DCNNs.

The forward pass of a convolutional layer can be described with the convolution operator $*$ analogous to 1.25 and 1.24. k denotes the k-th filter in layer l.

$$\mathbf{z}_k^{(l+1)} = (\mathbf{W}_k^{(l)}) * \mathbf{a}^{(l)} + \mathbf{b}_k^{(l)} \tag{1.32}$$

$$\mathbf{a}_k^{(l+1)} = f(\mathbf{z}_k^{(l+1)}) \tag{1.33}$$

The vectorized partial derivatives for gradient descent can then be computed like the following.

$$\begin{aligned} \nabla_{\mathbf{W}_k^{(l)}} J(W, b; \mathbf{x}, \mathbf{y}) &= \sum_i (a_i^{(l)}) * {\delta_{\mathbf{k}}}^{(l+1)} \\ \nabla_{\mathbf{b}_k^{(l)}} J(W, b; \mathbf{x}, \mathbf{y}) &= \sum (\delta^{(l+1)}) \end{aligned} \tag{1.34}$$

The sum of the second equation adds the entries of the error matrix $\delta^{(l+1)}$ [9].

1.3.3 Max-Pooling Layer

The *max-pooling* layer performs downsampling of a given input. It preserves the largest input and discards the rest. In image processing DCNNs it is usually placed after each convolutional layer reducing the width and height of the layer output [12], [13].

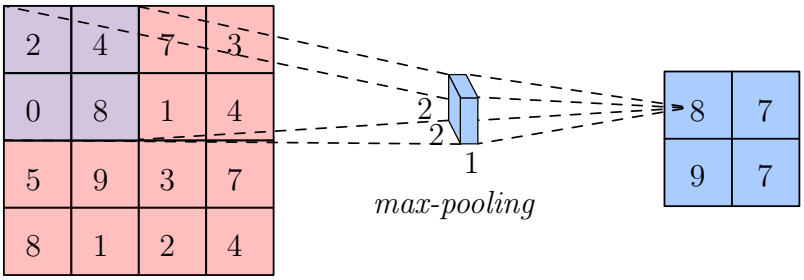


Fig. 1.12: Schematic drawing of a 2×2 sized *max-pooling* layer downsampling an 8×8 input to 4×4 with a stride of 2 so that no overlapping occurs.

During training the preserved outputs have to be stored, so that during backpropagation only the weights, which contributed to this output, are upgraded. The gradient through the discarded outputs is set to zero then.

1.3.4 Rectified Linear Unit (ReLU)

Although the Sigmoid function (s. Figure 1.6) has the advantage of better approximating the real behavior of neurons regarding a maximal limit of stimulation frequency, it can lead to disadvantages in ANNs. The saturating behavior can result in dying gradients during backpropagation. All output values are close to 0 or 1, if the neuron output is smaller than -4 or larger than 4. If e.g. the initialization pushes a neuron output to large values, any small weight update changes have a gradient close to zero, so that eventually no effective update is performed anymore and the learning process of this neuron is stuck. Less important but still worth mentioning is the computation time. 1.35 and its derivative are very efficient to compute. The sigmoind function 1.18 and its derivative 1.19 are more expensive to evaluate than the simple linear parts of a ReLU.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1.35}$$

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1.36}$$

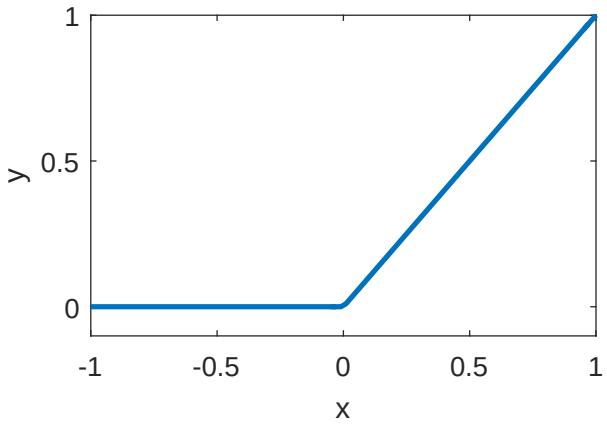


Fig. 1.13: ReLU function.

It also does not saturate for any positive neuron output and leads to faster convergence of DCNNs [12]. A modification of the standard ReLU is the leaky ReLU, which has a small slope $a < 1$ in negative regions to remedy the problem of saturation.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{otherwise} \end{cases} \tag{1.37}$$

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ a, & \text{otherwise} \end{cases} \tag{1.38}$$

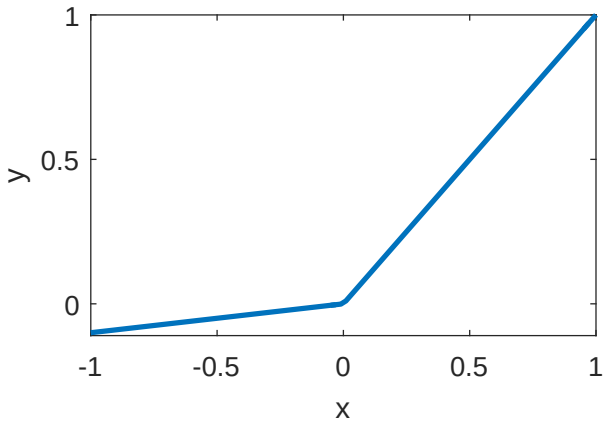


Fig. 1.14: Leaky ReLU function with parameter $a = 0.1$.

There exist several more modifications and activation functions [14] but ReLUs are a common choice in image processing because of the faster convergence and simple computation [12], [3], [4].

1.4 Evaluation Metrics

This section gives a short introduction of the evaluation units to measure the performance of the chosen approaches in this work. The general aim is to predict regions of interest (RoI). When extracting regions of interest for localization, a positive or negative result cannot be directly measured. To have a comparable measure of a successful or false predictions the intersection over union (IoU) of a predicted and the ground truth bounding box can be computed.

1.4.1 Intersection over Union

IoU is a way of measuring how similar the predicted and the ground truth bounding box are. Like the name suggests, it divides the intersection area by the union area. If A_p and A_t are the areas of the predicted and ground truth region of interest, the IoU can be calculated as follow.

$$\text{IoU} = \frac{A_p \cap A_t}{A_p \cup A_t} \tag{1.39}$$

[15] proposes an IoU value of 0.5 for a positive detection. The same criterion was adopted by the sources [16] and [3], whose proposed object localization methods are used in this work.

1.4.2 Precision and Recall

Now that a measure for positive and negative predictions exist it is possible to determine how well a localization system detects positive instances and how precise these predictions are. Namely, these values are recall and precision.

Considering a simple document search system with only two types (relevant and irrelevant), each document of a set is either relevant (positive) or irrelevant (negative). After a search is performed, every document in the set belongs to one of the following four different cases, when compared to the underlying ground truth.

- True Positive (TP): A relevant document is retrieved.
- True Negative (TN): An irrelevant document is rejected.
- False Positive (FP): An irrelevant document is retrieved.
- False Negative (FN): A relevant document is rejected.

This can be illustrated like in the following figure based on [17].

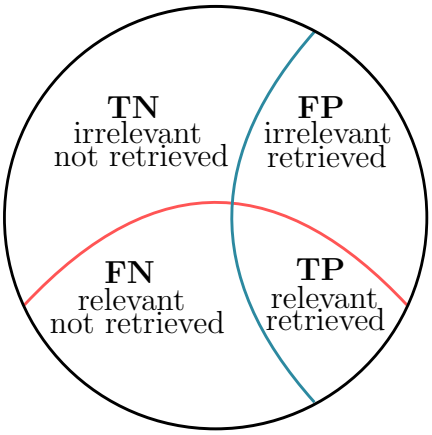


Fig. 1.15: The red line separates relevant items from irrelevant items. The blue line separates retrieved items from not retrieved ones. The abbreviations denote the four different cases of the list above.

More generalized, recall and precision can be defined like the following:

Recall measures the ability of retrieving items, that are relevant [17].

$$\text{recall} = \frac{\text{No. retrieved relevant items}}{\text{Total No. relevant items in data base}} = \frac{|\text{TP}|}{|\text{TP} \cup \text{FN}|} \tag{1.40}$$

Precision measures the ability of rejecting items, that are irrelevant [17].

$$\text{precision} = \frac{\text{No. retrieved relevant items}}{\text{Total No. retrieved items}} = \frac{|\text{TP}|}{|\text{TP} \cup \text{FP}|} \tag{1.41}$$

Examining both formulas leads to two observations:

1. If the number of false negatives is low and the number of true positives is not zero, the recall is high. → The system has a high recall, if it classifies every item as relevant.

2. If the number of false positives is low and the number of true positives is not zero, the precision is high. → The system has a high precision, if it classifies one relevant item as relevant and rejects the rest of the data set as irrelevant.

Thus only precision and recall together can be used to measure the performance of a system. A good system should have high precision and high recall [17].

One additional important note has to be made for the further evaluation. Since it is natural, that there are situations without any visible relevant object the image sets contain such scenarios. Recall and precision are computed for each image and then averaged over the respective set. To reward the system for rejection of false positives the precision is 100 %, if there is no object instance and no prediction. Otherwise it is 0 %. An empty image in the sense of no present object instance does not yield any information about the system's recall ability. Therefore, these instances are excluded for recall calculation.

2 Applied Methods for Object Localization

This chapter introduces two different approaches for object localization on images, which are later on compared in their performance. The first section points out constraints, which lead to the selection of both methods. Afterwards, both methods are explained in terms of their basic principles and implementation. Since the selected DCNN approach is oversized for the posed problem of this work, the proposed network structure is scaled down. This results in four problem adjusted network structures, which will also be evaluated among each other in chapter 4.

2.1 Constraints

Since the Nao is supposed to perform autonomous actions and decisions it is limited to the built-in hardware. Additionally as stated in the previous chapter, the performance of any machine learning technique relies on the provided training data. Both constraints will be explored in the subsequent sections.

2.1.1 Hardware

Images are usually stored in an array or a more sophisticated type of data structure, which can store the color model information and position of each pixel. When processing images for object localization, typically patches of images are processed instead of single pixel values. This results in multiple vector product computations and leads to the hardware related limitation of floating point operations per second (FLOPS). The Intel Atom Z530 can perform approximately 1 GFLOPS using vector products on a Linux based operating system [18]. The Nao's frame rate is set to 30 frames per second (FPS). Thus every computation should be completed in a time frame of approximately $\frac{1}{30}$ seconds. This leaves $\frac{1}{30} \cdot 1$ GFLOPS $\approx$ 33.3 MFLOPS for one cycle, which is shared by all other modules running simultaneously. The number of floating point operations of the implemented approaches should therefore be kept as low as possible and never exceed 33.3 MFLOPS. If the number of available floating point operations per cycle is exceeded, it cannot be guaranteed anymore that all modules running can finish their respective calculations in time. This in turn can lead to a complete failure of the system, if e.g. the necessary computations for stable walking cannot be performed in time.

2.1.2 Training Data

The diversity and amount of training data is crucial for the performance of machine learning methods but also depends on the machine learning system itself. If it has many free adjustable parameters, it can reflect complex relationships and decisions. However, numerous free parameters require huge amounts of training examples, to avoid overfitting which will be elaborated in more detail in the following chapter. Consequently the time required for one epoch (i.e. the system has trained with each example once) of training

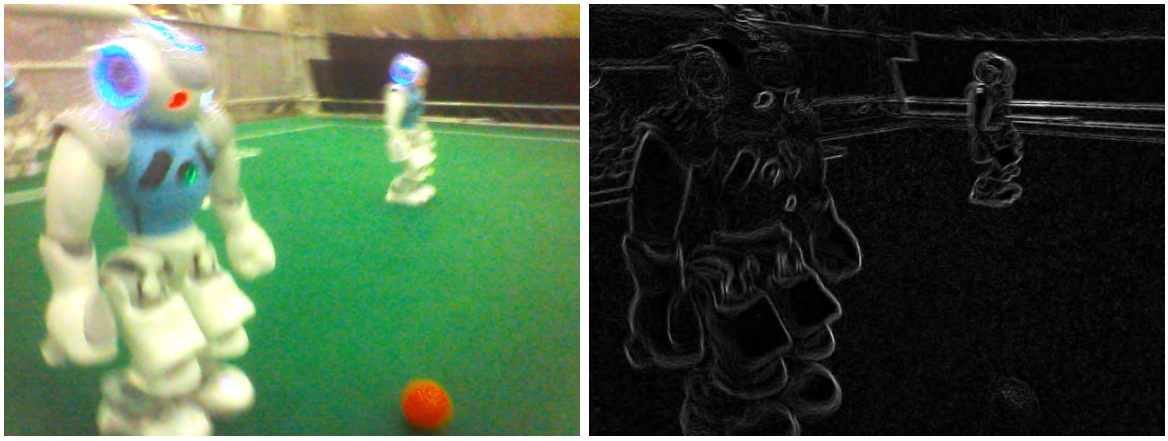
rises with the number of training examples. This again makes it more time consuming to perform system validation to tune manually adjustable parameters. Especially neural networks require multiple validations because they can settle in a local minimum. Thus training time must also be taken into consideration because it can be a limiting factor.

To guarantee a diverse set of training data, the data base for training, validation and testing contains images taken by the Nao during robot soccer events in Brazil, China and Hamburg. It consists to one part of images taken in-game, which can include instances of robots, balls and goals but also none (i.e. background). The other part consists of staged situations with different lighting conditions.

So far there is no application for automated generation of images with tagged regions of interest for SPL robot soccer, making huge amounts of labeled training data hardly accessible. The labeling had to be performed manually and resulted in a base set of approximately 1700 images.Furthermore, data augmentation techniques were applied to extend the data base by introducing more variation in terms of position and scale. These methods are explained and evaluated in chapter 3 and 4.

2.2 Contrast Based Object Localization

This approach focuses on a human selected image feature. Selecting image features in machine learning offers the benefit of more insight and control over the learning and is therefore less prone to overfitting. Depending on the application there can be useful features, which are robust against variation of certain environmental conditions, which may require a lot of training examples to learn. In the case of object detection, objects can be seen as stand-alone things with closed boundaries and centers [16]. A gradient image depicts the change of intensity or color in an image. Combining these properties leads to the conclusion, that normed gradient (NG) features of an image are a good discriminant for object detection because little variations of closed boundaries do not have a huge impact on their representation.



a) Example image. b) NG map of example image.

Fig. 2.1: Original image taken in-game by a Nao robot (left) and the corresponding NG map.

Figure 2.1 shows a natural image and the corresponding image of NG features. NG features only represent intensity change without information about its direction, which results in a more compact representation to make calculations more efficient. NG features are also insensitive to changes of translation, scale and aspect ratios, making them a robust feature.

[16] proposes an approach using binarized normed gradients (BING) for object detection. The abbreviation BING will be used to denote this approach from this point on. BING uses two cascaded SVMs and binary model approximation to speed up feature extraction and testing. [16] claims to generate object proposals at 300 FPS and being approximately 1,000 times faster than comparable methods in their test set up (CPU: Intel i7-3940XM) making this fast approach a suitable candidate for object localization on the Nao robotic system. The cascaded SVMs have to learn 136 free parameters in the proposed configuration and were trained with the PASCAL VOC 2007 image set, which contains roughly 5,000 training images and 20 different object classes. In this work the SVMs are trained with nearly 30,000 images, which should be enough to avoid overfitting, although they are mainly augmented and not original.

2.2.1 Implementation and Methodology

The complete implementation is available online as C++ code and was performed by the authors of [16]. Consequently the following implementation details are based on [16].

The core idea of BING is to use a sliding window approach and two cascaded SVMs to localize objects on an image. Before any localization is performed, the image is transformed into a map of NGs. This is done by applying a one dimensional mask $[-1, 0, 1]$ to compute the horizontal and vertical image gradients g_x and g_y. The NG map is then calculated with $\min(|g_x| + |g_y|, 255)$ and saved in BYTE values. The sliding window has a fixed size of 8×8 and is fed to the first SVM making the window a 64 dimensional NG feature. To capture objects of different sizes and shapes the 8×8 window scans over predefined

quantized window sizes. I.e. the NG map is reshaped to 36 different scales and aspect ratios as depicted in the following figure.

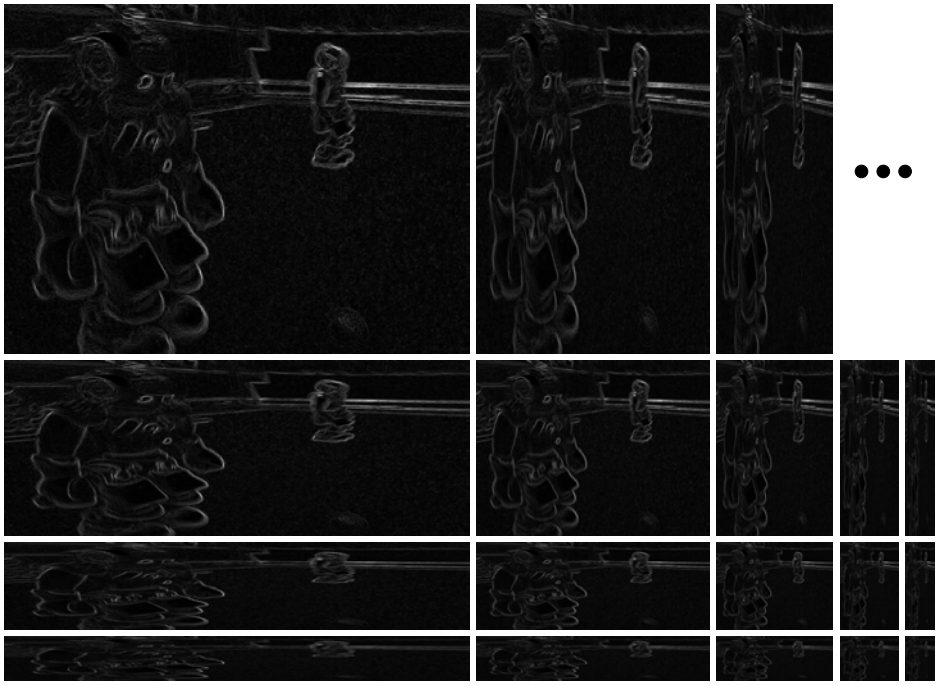


Fig. 2.2: Exemplary quantized windows of a NG map with a quantization base of $q = 2$. It halves width and height in each step.

Each window $\mathbf{g}_l$ is scored with the learned linear model $\mathbf{w} \in \mathbb{R}^{64}$.

$$s_l = \langle \mathbf{w}, \mathbf{g}_l \rangle \tag{2.1}$$

$$l = (i, x, y) \tag{2.2}$$

s_l is the filter score of an NG feature window $\mathbf{g}_l$ at a certain location l. A location l is defined by the quantized windows size i and the corresponding position of the sliding window on the NG map specified by x and y. After applying non-maximum suppression (NMS) to each quantized window size i a small set of proposals is left for each i. NMS is a method to remove redundant bounding box proposals. Redundant bounding boxes are basically multiple proposals for the same region and only differ slightly by their position and size. To measure the "redundancy" the IoU value between boxes can be calculated. A basic approach, which is also used in BING, sorts all boxes by its score in descending order and compares the highest scoring proposed bounding box to the rest. All proposals, which have too much overlap in terms of e.g. IoU, are discarded. In the next step the iteration starts from the following highest scoring box, which has not been discarded by the previous steps [19]. Since NMS is applied for each quantized window separately, the IoU value does not have to be computed. Each prediction results from the sliding 8×8 window and thus position and size are known within one quantized window. After sorting

the proposals by their score, the nearest neighboring windows of the highest scoring one are discarded. The distance, which defines neighbors is a hyperparameter and set to 2 as default. Typically very narrow windows (e.g. 10×500) are less likely to contain an object instance compared to a quadratic window (e.g. 100×100). Thus a second filter score is defined, which is computed by the second SVM.

$$o_l = v_i \cdot s_l + t_i \tag{2.3}$$

v_i and t_i are coefficient and bias terms evaluating the filter score s_l based on the quantized window size i to obtain an objectness score o_l. Both SVMs are optimized with respect to the following problem [16] [20]:

$$\min_{\mathbf{w}} \|\mathbf{w}\|_1 + C \sum_t (\max(0, 1 - r^t \mathbf{w}^T x^t))^2 \tag{2.4}$$

2.4 is slightly modified to match the notation of chapter 1, with r^t and x^t as ground truth and training instance pairs of sample t and $\mathbf{w}$ as general hyperplane to be optimized. Additionally to the standard formulation the error is squared to penalize larger deviations more than smaller ones and a penalty term $\|\mathbf{w}\|_1$ is added. $\| \ \|_1$ is the 1-norm and penalizes the sum of weights, which yields a sparse solution $\mathbf{w}$ [20]. The learned linear model $\mathbf{w}$ can look like the following.

Fig. 2.3: Learned model $\mathbf{w}$ of [16].

Figure 2.3 shows large weights along the borders of $\mathbf{w}$. The large weights seem to separate an object in the center from its background. Because $\mathbf{w}$ is learned from real example instances, it is more sophisticated and realistic compared to manually designed weights. $\mathbf{w}$ puts for example more emphasis on upper body object boundaries. This makes sense because the upper body usually covers more enclosed area than legs and therefore contains more information about object presence.

To speed up the NG feature extraction and testing process a binarized approximation method is used, which is explained in more detail in [16]. It basically approximates the learned model $\mathbf{w}$ and the extracted feature window with binary values. These can ideally be stored by an INT64 variable which in turn explains the choice of an 8×8 window. BITWISE operations like OR, ADD or SHIFT can then be used to test BING features.

2.3 Deep Convolutional Neural Network Object Localization

DCNNs typically have a less controlled approach. They are often used in an end-to-end learning scheme, in which a training example is presented and the desired output is compared to the real output for error backpropagation and weight updates. The filters for feature extraction are learned during training and not predefined. This offers the benefit of little human interaction and might lead to extraction of features, which are not plausible for humans but effective. However, the lack of control also poses disadvantages. Since the feature extraction filters are learned from training data, their robustness to change in environmental conditions is highly reliant on the training data set. A diverse and large set of training images is therefore necessary to avoid overfitting.

Many DCNN approaches for object localization involve a preceding region of interest proposal generator (e.g. *R-CNN* [21]) or consist of extremely large network structures (e.g. *DeepMultiBox* [22]). The chosen method for this work is based on [3]. The DCNN of [3] is claimed to be among the fastest performing real-time object detectors and operates at 155 FPS in the fastest configuration. It is a simple feedforward network consisting only of convolutional, downsampling and fully connected layers. The system models object detection as a regression problem to localize and classify 20 different object classes. It divides the input image into an even grid through convolution and downsampling and predicts bounding boxes and confidence scores for each box.

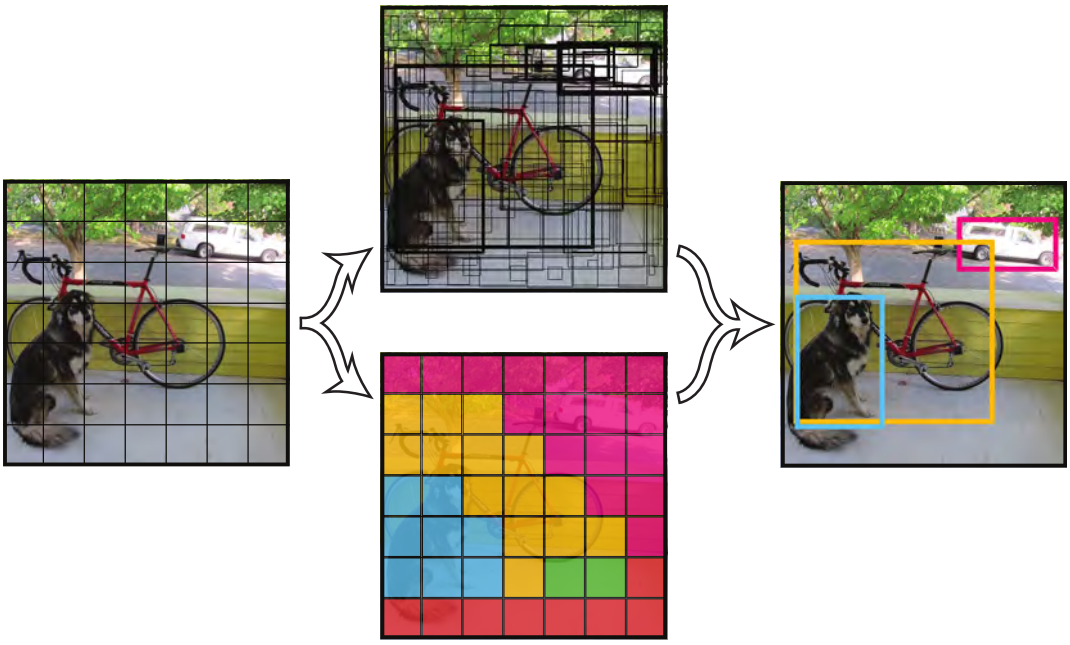


Fig. 2.4: Illustration of how spatial information is preserved through the computed image grid. Two bounding boxes are predicted for each grid cell. The highest scoring bounding boxes are shown on the right side [3].

The training set consists of approximately 1,210,000 images taken from the *ImageNet* 1000-*class* and *PASCAL VOC* 2007 and 2012 competition data sets [2], [23], [1]. Since it

contains 32,650,000 free parameters with only 1,210,000 training images, extensive data augmentation is applied during training. However, the used hardware (NVIDIA GeForce Titan X), GPU computation and network depth (appr. 32,650,000 free parameters) exceed the previously mentioned constraints by far.

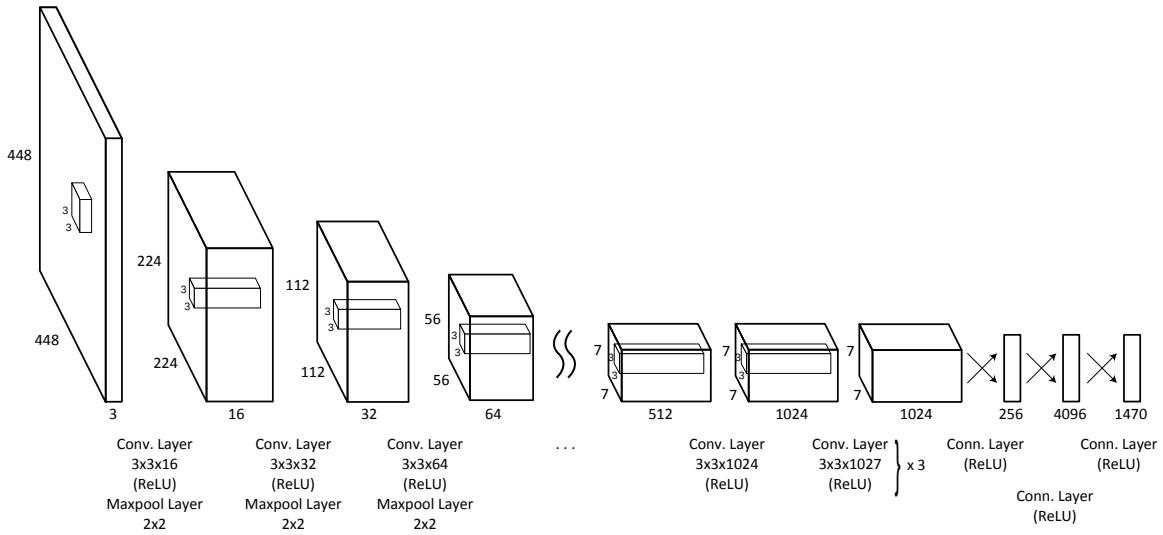


Fig. 2.5: Network structure of [3] in its fast configuration.

Figure 2.5 shows the described network. The large number of free parameters also indicate a large number of necessary floating point operations which have to be reduced heavily to run on a real time system like the Nao robot. In addition the roughly 800 available original training images are not suitable for 32,650,000 free parameters even if extensive data augmentation is applied. Hence free parameters and thus the number of layers should be minimized to avoid overfitting and fulfill the hardware constraints. A more detailed view and explanation of the implemented minimized network structure is provided in the next section.

2.3.1 Implementation and Methodology

This section describes the problem adjusted and downsized network implementation. However, for a better understanding, the original network in its fast configuration is described first. The description in this section is based on [3].

The general structure is given in Figure 2.5. It consists of a repeated convolution and *max pooling* scheme to extract features and create a 7×7 map of deep features. Each hidden layer has a leaky ReLU as non-linear activation function. Only the output layer uses a simple linear activation because real values should be predicted and thus negative or too large ones should lead to a higher error value. The network predicts 1470 output values. They consist of 20 class specific scores, 2 bounding box locations (x, y, $\sqrt{w}$, $\sqrt{h}$) and 2 corresponding objectness scores for each grid cell. The square root of w and h should reflect that small errors in small bounding boxes matter more than small errors in large bounding boxes. The true bounding box width w and height h are normalized by the

image width and height, so that they fall between 0 and 1. The bounding box location defined by x and y is parametrized to be an offset of a particular grid cell and is also bound between 0 and 1. If $x = 0$ and $y = 0$, the bounding box center of a grid cell would lie in the upper left corner of that particular grid cell. If $x = 1$ and $y = 1$, the bounding box center would lie in the lower right corner of that grid cell. A grid size of 7×7 yields:

$$\begin{array}{ccccc} & & \text{class scores} & & \text{outputs} \\ (\underbrace{7 \times 7}) & \times & (\overbrace{20} & +2 \times (\underbrace{1+4})) & = \overbrace{1470} \\ \text{grid size} & & \text{objectness score} & + \text{location} & \end{array}$$

The loss function looks like the following.

$$\begin{aligned} Err = & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2 \end{aligned} \tag{2.5}$$

S denotes the grid size and B the number of bounding box predictors per grid cell. Thus index i defines one specific grid cell and j a particular predictor. x_i, y_i, w_i and h_i are the predicted bounding box locations and sizes. To reflect that small deviations in small bounding boxes matter more than small deviations in large bounding boxes the predicted square roots of $\hat{w}_i$ and $\hat{h}_i$ are incorporated in the loss function. C_i is the network confidence of object presence and $p_i(c)$ the class specific probability score for one of the classes $c \in$ classes. The respective hatted values $\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i, \hat{C}_i$ and $\hat{p}_i(c)$ are ground truth values to compute the sum squared error. $\mathbb{1}_i^{\text{obj}}$ is either 0 or 1 and denotes if an object is present in grid cell i. $\mathbb{1}_{i,j}^{\text{obj}}$ is 0 or 1 and denotes the "responsible"predictor of an object. Only the predictor, which predicts the bounding box with the highest IoU value for an object, is marked as "responsible". This leads to specialization among all predictors. Each predictor improves at predicting certain sizes, aspect ratios or classes of objects. $\mathbb{1}_{i,j}^{\text{noobj}}$ is 1, if no object is present in grid cell i and 0 otherwise. Thus only the confidence for an object in grid cell i is penalized, if no object is contained in it. Lastly there are λ_{coord} and λ_{noobj}, which are adjustable parameters. λ_{coord} weights the localization error and λ_{noobj} the confidence error, when no object is present. Often the major number of grid cells

do not contain an object, which pushes the confidence scores of these cells towards zero. This again can overpower the gradients of cells containing an object leading to model instability. Therefore the localization error is weighted higher than the confidence error ($\lambda_{\text{coord}} = 5$, $\lambda_{\text{noobj}} = 0.5$).

During real application, NMS is applied to remove redundant bounding boxes, which depict the same object. After sorting all predicted boxes by their respective confidence score, the IoU value between the highest scoring box and each subsequent box is computed. If the IoU value exceeds 0.55 the lower scoring box is removed. This process iterates through all non-discarded boxes.

The basic idea of the described approach is kept when downsizing the DCNN. I.e. the network stays a feedforward network, which needs one evaluation to produce localization proposals and confidence scores and it divides the image into an even grid through convolution and downsampling (s. Figure 2.4 and 2.5 for illustration of downsampling and grid). To accomplish this goal, the network is downsized evenly in a sense that the input and output sizes as well as the layer numbers and layer depth are reduced. The loss function is also scaled with regard to the simpler problem and limited computational power.

To predict scores and locations, whilst taking the spatial information of the grid into consideration, at least one fully connected layer is necessary as final layer. Subsequently every output value results from a dot product of a weight vector and the complete previous grid layer. This again shows that the number of outputs and its preceding layer have a large impact on the number of free parameters. It is therefore especially important to scale the fully connected layer and the preceding one down. Since this work only deals with Nao robots the 20 class specific scores fall off. The grid size is reduced to 3×3. Lastly each grid cell only predicts one bounding box because each additional predictor would introduce five more output values for each cell.

$$\overbrace{(3 \times 3)}^{\text{grid size}} \times \underbrace{(1 + 4)}_{\text{robot score + location}} = \overbrace{45}^{\text{outputs}}$$

Compared to the original network output illustration (s. Figure 2.4) the downsized version looks like the following.

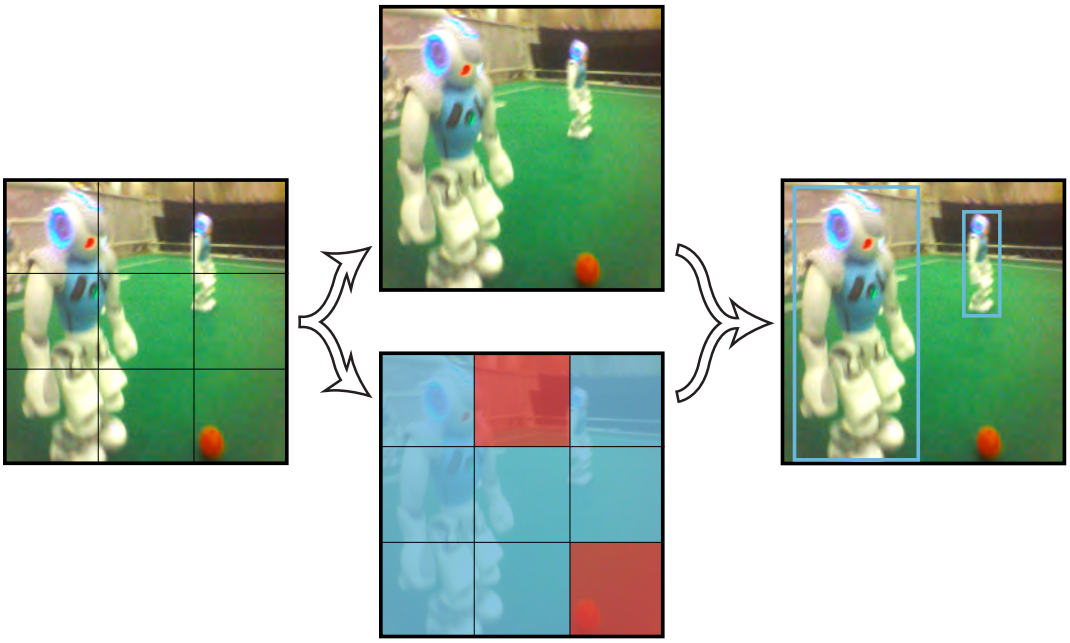


Fig. 2.6: Illustration of downsized image grid comparable to Figure 2.4. Only one bounding box is predicted for each grid cell. Larger grid cells can lead to less precise localization and small objects might be undetected.

The figure above illustrates the 3×3 prediction grid projected onto the original input image. It is clearly visible that a coarser grid will have more difficulties to localize small objects, looking at the upper right grid cell of Figure 2.6. Only a small fraction of this cell is covered by the robot, which could likely yield a false localization or even no detection of the robot presence at all. However, increasing the grid size from 3×3 to 4×4 would already yield 80 outputs and thereby almost double the number. This would highly increase the number of free parameters. The influence of the final fully connected layer on the parameter number can be seen later in this section in Table 1. The error function can be rewritten as follows.

$$\begin{aligned} Err = & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \left(C_i - \hat{C}_i\right)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2 \end{aligned} \tag{2.6}$$

To utilize the benefits of a DCNN at least two convolutional layers are necessary, such that

features of a combination of features are extracted. Based on the fact that every additional layer introduces more floating point operations and parameters, the hidden layer number is set to two. This allows more different testable setups since one full training needs less time and thus gives a better basic insight. The downsampling process from input layer to a 3×3 grid should be as even and smooth as possible, i.e. the input of each hidden layer is downsampled by the same constant factor and should not be too drastic (e.g. from a 400×400 layer input to a 10×10 layer output). Furthermore there is a trade-off between input size and necessary floating point operations. However, the input size must not be too small, otherwise important information cannot be detected anymore. If, for example, an image is downsampled from 400×400 to 10×10 in the input space, the chance of detecting and localizing an object are extremely low because too much fine grained information was lost. Since there is the chance of multiple robots, which do not necessarily cover the whole image the input resolution of the image should not be below 100×100 pixels. A squared image dimension is chosen for simplicity of the implementation. Otherwise the layer outputs would also differ in their aspect ratio and introduce more complications when further downsampling by the network is performed. Because the input image is scaled down before being fed to the network, fine grained information is lost in any case. Thus resizing from a ratio of $4 : 3$ to $1 : 1$ only introduces a relatively small distortion and is generally performed on all input images. To realize a reasonable input resolution and a quick downsampling to a 3×3 grid after two hidden layers the downsampling is performed by both convolutional layers and *max-pooling* layers. The stride of the convolutional filter is therefore set to three pixels instead of the common choice of one. If a stride of one is chosen, parts of image patches are evaluated several times, which leads to high computational effort. The *max-pooling* layers have a stride of two pixels and a size of 2×2 pixels. Lastly the size of the convolutional filters should be discussed. To keep the number of filters small it is recommendable to choose filters with a height and width larger than three pixels. Thus, the filters can learn few coarse features instead of a combination of many small filters, which need more training data to prevent co-adaption symmetric learning. Since a final 3×3 grid is required, the input width and height may vary depending on the filter sizes.

The general structure of the designed network is depicted in Figure 2.7 below.

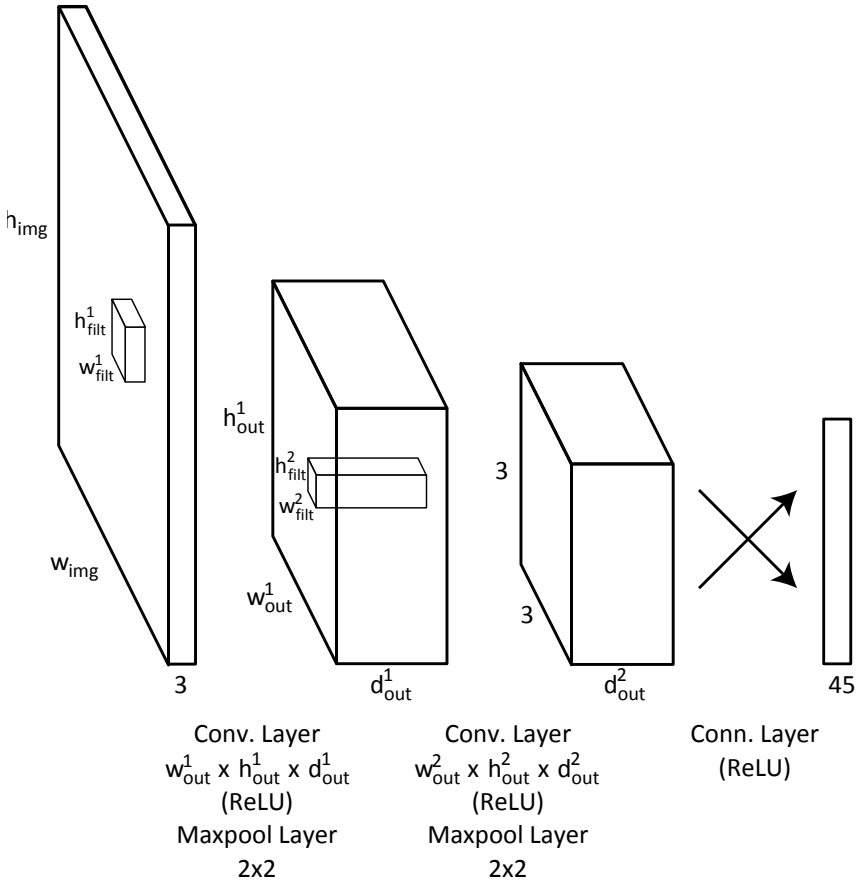


Fig. 2.7: General network design. An appropriate filter size and number will be determined below.

The filter sizes $w_{filt}, h_{filt}, d_{filt}$ and layer depths d_{layer} are not specified because in this setup the depth of a filter always equals the depth of its input. Which leads to the fact that the number of filters in each convolutional layer also influence the number of free parameters in the subsequent network layers. The following formulas can be used to calculate the number of free parameters and necessary floating point operations for one feedforward evaluation.

Let NoP_{filt} be the number of parameters of a convolutional filter, w_{filt} and h_{filt} be the width and height of a convolutional filter (e.g. 3×3) and d_{inp} be the depth of the input to a convolutional filter, then the number of free parameters of a filter NoP_{filt} can be calculated like the following.

$$NoP_{\text{filt}} = w_{\text{filt}} \cdot h_{\text{filt}} \cdot d_{\text{inp}} + 1_{(\text{bias})} \tag{2.7}$$

If several identically dimensioned filters in a layer process the same inputs, the number of free parameters per filter NoP_{filt} is multiplied by the number of filters NoF_{layer} in that layer because each filter possesses its own trainable set of parameters.

$$NoP_{\text{conv-layer}} = \left(w_{\text{filt}} \cdot h_{\text{filt}} \cdot d_{\text{inp}} + 1_{(\text{bias})}\right) \cdot NoF_{\text{layer}} \tag{2.8}$$

The number of free parameters of a fully connected layer can be calculated using the number of outputs $NoO_{\text{fc-layer}}$ and the dimension of the input.

$$NoP_{\text{fc-layer}} = \left(w_{\text{inp}} \cdot h_{\text{inp}} \cdot d_{\text{inp}} + 1_{(\text{bias})}\right) \cdot NoO_{\text{fc-layer}} \tag{2.9}$$

Let $NoO_{\text{conv-layer}}$ be the number of outputs of a convolutional layer, then the number of floating point operations for one evaluation for that layer $NoFLOP_{\text{conv-layer}}$ can be calculated as follows.

$$NoFLOP_{\text{conv-layer}} = w_{\text{filt}} \cdot h_{\text{filt}} \cdot d_{\text{inp}} \cdot NoO_{\text{conv-layer}} \cdot 2 \tag{2.10}$$

Every output value is computed by a dot product of the filter with an input patch. Thus $w_{\text{filt}} \cdot h_{\text{filt}} \cdot d_{\text{inp}}$ multiplications and $w_{\text{filt}} \cdot h_{\text{filt}} \cdot d_{\text{inp}} - 1$ summations are necessary. Eventually the bias term of that filter is added resulting in $w_{\text{filt}} \cdot h_{\text{filt}} \cdot d_{\text{inp}} \cdot 2$ floating point operations for each output value.

The number of necessary floating point operations for a fully connected layer $NoFLOP_{\text{fc-layer}}$ with a given output number $NoO_{\text{fc-layer}}$ can be calculated in similar fashion. A fully connected layer is then basically a filter with an equal shape of its input.

$$NoFLOP_{\text{fc-layer}} = w_{\text{inp}} \cdot h_{\text{inp}} \cdot d_{\text{inp}} \cdot NoO_{\text{fc-layer}} \cdot 2 \tag{2.11}$$

The following table presents different network layouts regarding filter number and size, the corresponding number of free parameters and the necessary floating point operations for one feedforward evaluation.

Filter Size and Number, Hidden Layer 1	**Filter Size and Number, Hidden Layer 2**	**Free Network Parameters**	**FLOP per Evaluation**
Filt. size: 5×5 Filt. number: 2	Filt. size: 5×5 Filt. number: 4	$356_{\text{conv}} +$ 1665_{fc}	$123600_{\text{conv}} +$ 3240_{fc}
Filt. size: 5×5 Filt. number: 4	Filt. size: 5×5 Filt. number: 8	$1112_{\text{conv}} +$ 3285_{fc}	$254400_{\text{conv}} +$ 6480_{fc}
Filt. size: 7×7 Filt. number: 2	Filt. size: 7×7 Filt. number: 4	$692_{\text{conv}} +$ 1665_{fc}	$291648_{\text{conv}} +$ 3240_{fc}
Filt. size: 7×7 Filt. number: 4	Filt. size: 7×7 Filt. number: 8	$2168_{\text{conv}} +$ 3285_{fc}	$597408_{\text{conv}} +$ 6480_{fc}

Tbl. 1: Overview of different network layout configurations and respective parameter and floating point operations count (FLOP denotes in this table the number of floating point operations).

Table 1 shows, that the dimensions of the fully connected layer in combination with the last convolutional layer have a high impact on the parameter number, whereas the dimensions of the convolutional layers mainly increase the floating point operations. The different setups will be evaluated in chapter 4.

2.4 Summary

Based on the defined general hardware constraints for methods, which are supposed to run on the Nao robot, two object localization methods were selected and evaluated with regard to these. Furthermore the principles and implementation details of the selected approaches were explained. The original proposed DCNN structure of [3] had to be adjusted to the mentioned constraints and resulted due to a variety of options in four final network structure proposals, which will be further evaluated in chapter 4.

3 Training

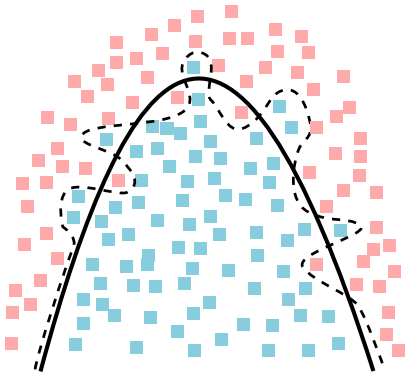
Training is the essential part of any machine learning method because in this stage the actual learning takes place. This chapter discusses problems arising during training and options to resolve these.

3.1 Difficulties

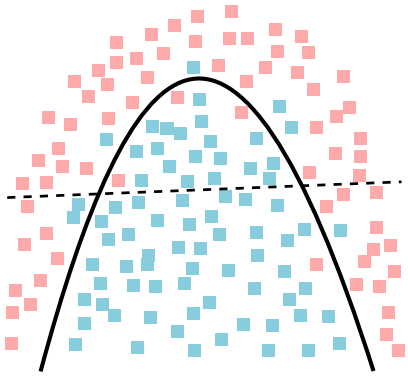
Two major problems in machine learning are overfitting and necessary training time. Overfitting occurs when insufficient amounts of training data is available for the number of trainable parameters. Training time rises, when there are a lot of trainable parameters and respectively lots of training data to be processed. Both problems will be explored in the next sections.

3.1.1 Overfitting

Finding the appropriate machine learning method and complexity of a system to approximate the desired function can be problematic. Especially for ANNs there exists no theoretical model to determine the appropriate number of neurons for a given data set and the desired performance. Generally, an increase of model complexity leads a to an increase of accuracy during training because the model is very "flexible" and fits the training data very accurately. If it is applied to unknown data, it fails because it also "remembers" unimportant features like noise as important to perform better on the training data. This process is known as overfitting.



a) Two class classification problem with an overfitting (dashed) and the true (bold) underlying decision boundary.



b) Two class classification problem with an underfitting (dashed) and the true (bold) underlying decision boundary.

Fig. 3.1: Both figures illustrate a two class (blue and red) classification problem. The underlying true decision boundary is depicted by the bold line. Blue and red instances, which lie on the wrong side are outliers, which result from random noise.

If the system is less complex it has a better generalization ability because it cannot fit the noise. But a too simple system will underfit and will not be able to represent the desired but unknown function.

The dilemma between complex systems, which could possibly overfit and simple systems, which could in turn underfit, is known as the *bias-variance dilemma* [24].

3.1.2 Necessary Training Time

In the case of SVMs a complex system would consist of a large input or feature space if the Kernel-Trick is applied. If the input is e.g. 1000-dimensional, the optimal separating hyperplane in the 1000-dimensional input space has to be found by optimization methods subject to the constraints in section 1.2.1. In case of an ANN the complexity can be determined by the number of neurons and overall trainable parameters in the network. The training of a feedforward network is performed through error backpropagation and iterative weight updates. The more parameters there are, the more computational effort goes into error backpropagation. Independent of the selected machine learning method a complex system needs a large and diverse set of training examples to negate overfitting. Thus the training needs more time until it has seen every training example at least once and the training process takes longer because the optimization has to be performed on a larger set of parameters.

Most machine learning methods also include tunable parameters to adjust the influence of certain error values and constraints. Finding the right setting can take several training and validation runs which is highly time consuming, if one full training takes several days or even more. In contrast to SVMs, ANNs do not pose a problem which can be solved by convex optimization. Gradient descent algorithms can get stuck in local minima during optimization. To decrease the risk of this case several iterations of random weight initialization and training can be applied, which makes training time again a crucial factor.

3.2 Solutions

To overcome overfitting the following solutions can be considered. Increased training time due to complex models is more of an unavoidable but considerable effect, which should be taken into account during the method complexity design stage.

3.2.1 Validation and Early Stopping

An overfitted system performs well on the training set but fails on unseen data. I.e. the training error will converge to zero during training but will be very high, when the system is exposed to unseen data. Thus the training process should be stopped, when overfitting starts. To determine when a system is starting to overfit, a test on unseen data can be performed in constant intervals during training. This constant testing is called validation. The validation set consists of data, which has been previously taken out of the training

set. A more detailed view of the training and validation set and their contents is provided in 3.3.

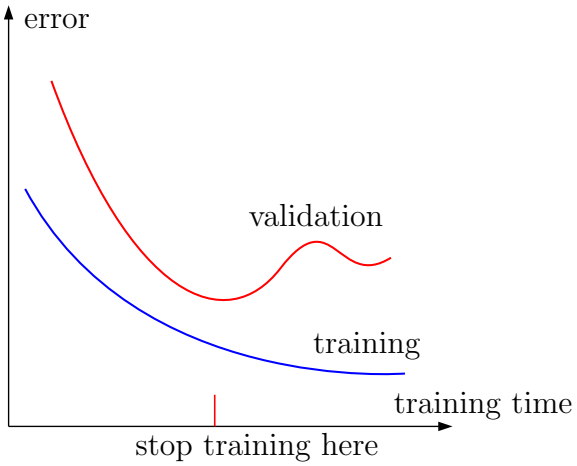


Fig. 3.2: Error curve of training and validation set with ideal stopping point for training [17].

Figure 3.2 illustrates the error curve with respect to the training validation set. When the validation error reaches its minimum and diverges training should be stopped. This procedure is called *early stopping*.

3.2.2 Weight Decay

An overfitted system shows its adaptation to the training data in highly specialized parameters. This specialization and imbalance of parameter values results from an incomplete training set, which leads the learning algorithm into these parameter updates. Taking a classification problem as example, the outcome can look like Figure 3.1 a). The strong influence of a few parameters leads to a very specialized and thus overfitted decision boundary.

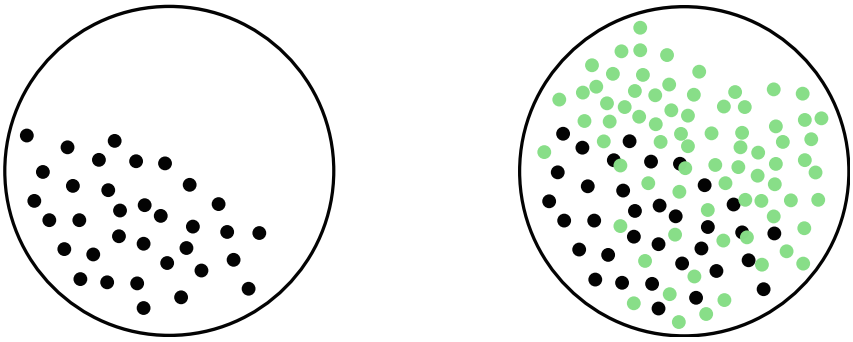
This characteristic of large specialized parameters can be utilized to counter overfitting during training. An extra term is added to the cost function J [24], [9]:

$$J^* = J + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(w_{ji}^{(l)} \right)^2 \tag{3.1}$$

This extra term sums all squared weight values and is scaled by the hyperparameter λ. This penalizes especially a sparse distribution of large weight values because of the square term. It thereby prefers simple solutions over complex ones. The choice of λ has to be determined heuristically because it depends on the number of parameters and the problem itself. If λ is too small, it has almost no effect on the cost function, if it is too big, it decreases the weight values irrespective of the actual model cost function [24]. This form of regularization is also known as *Tikhonov* regularization or L^2 regularization.

3.2.3 Data Augmentation

With data augmentation techniques it is possible to artificially create or enlarge data sets. The aim is to create data instances, which seem like unseen data to the machine learning algorithm.



a) Relevant possible input combinations covered by training set.

b) Relevant possible input combinations covered by training set and augmented data.

Fig. 3.3: The circles depict the relevant input combinations, which might occur during application of machine learning algorithms. The black dots are the input instances covered by the training set. The green dots are the additional augmented data instances.

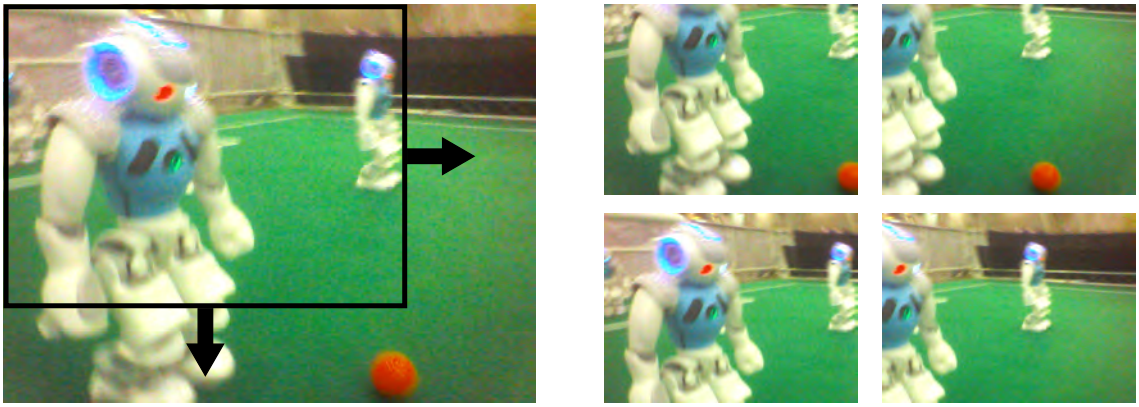
Figure 3.3 illustrates how data augmentation can improve robustness of a system by providing more and diverse data instances. This can be very useful for the task of object localization, which needs a certain level of complexity but insufficient amounts of training data are available. The following section presents the data augmentation techniques used for this work.

Data Augmentation Methods

[12] and [3] also utilize data augmentation to artificially enlarge training data and prevent overfitting. Both use image flipping, cropping and scaling as well as color channel modification.

Horizontal flipping of a base image changes the position and orientation of an object and is a very simple way to enlarge the set by a factor of two.

The idea of image section cropping is to vary the position of objects on images and also introduce a different scaling. A cropping window of a reasonable size is chosen (e.g. 75 % of the original image size), which should have the same height and width ratio as the original image, so that the generated data is not distorted.



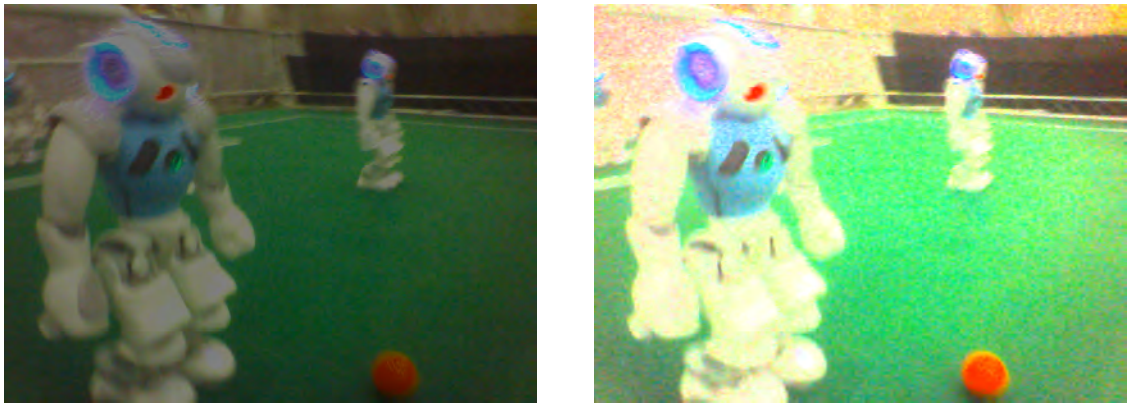
a) Sliding window moving over input image in depicted directions.

b) Resulting four image crops.

Fig. 3.4: The sliding crop area moves in steps of e.g. 80 pixels over the image and crops out image parts. The resulting parts can be seen on the right. The robot positions and scales differ, if the cropped parts are resized to the original image size.

Afterwards this window moves with an adjustable step size of several pixels over the image and crops image sections. Since the original labeling data is known, the bounding boxes for the cropped section can be computed and no manual labeling is necessary. Additionally this method also introduces a new scaling of each object because most methods require a fixed input size and therefore re-size an input of differing dimensions.

To further introduce robustness towards a change of lighting conditions [3] proposes random adjustments of brightness in the hue, saturation, (brightness) value (HSV) color model. This can be simply performed by transforming the training image into the HSV color model representation and scaling the V channel.



a) Example image after brightness value scaling with 0.5.

b) Example image after brightness value scaling with 1.5.

Fig. 3.5: Both images have scaled brightness values to simulate darker and brighter lighting conditions compared to the original 3.4.

There are several additional and alternative methods for image data augmentation, which could be utilized (s. [25]), but this work is limited to the applied methods of [3] because the basic DCNN structure is based on this source and it is reasonable to also adapt the augmentation methods.

Whereas image flipping and cropping are obviously improving the generalization ability of a DCNN to localize objects because the position of robots changes, it is not clear that color channel jittering significantly improves the generalization ability regarding different lighting conditions. This is evaluated in section 4.3.3 of the following chapter.

3.3 Data Base

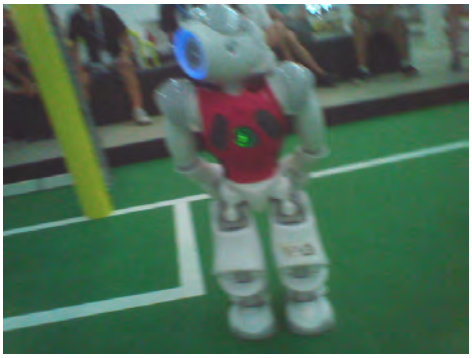
A well performing image processing system for object classification and detection for autonomous robot systems should be generally noise resistant and robust against changes of lighting conditions. Furthermore, it should output very few false positives because a positive usually leads to an action. This could be a movement, a decision or transfer of false information. In any case, the resulting action would lead to a worse situation e.g. avoiding obstacles, which do not exist. The training images should therefore contain information, which lead to robust and generalizable Nao robot model features.

The used data base of original images consists of 1748 images, which were taken by Nao robots with different camera parameters and in different locations. Therefore noise and different lighting conditions are introduced into the set, which makes the trained system more noise resistant and lighting independent. The locations include:

Set Location	**No. Images**	**Abbreviation**
Nao Robot Laboratory at TUHH	131	TUHH Lab
Nao Robot Laboratory at HTWK	20	HTWK Lab
Building N at TUHH	601	TUHH N
Robotic Hamburg Open Workshop at TUHH	262	RoHOW
RoboCup, Brazil	590	Brazil
RoboCup, China	144	China

Tbl. 2: Introduction of data base images.

The ratio of images containing a robot and background is 1:1. Images containing robots can have a single or multiple instances. Background images are either in-game scenes without a visible robot or scenes containing a room ceiling, walls etc.



a) Single robot instance.

b) Multiple robot instances.

c) Background image of in-game situation.

d) Background image of undefined situation.

Fig. 3.6: Exemplary image instances of different situations contained in the data base.

To further generalize the system, images of Nao robots contain Naos with blue, red or no jerseys. Unfortunately, there is a bias towards blue and red jerseys because the in-game footage can only be taken with Nao jerseys. Annotations, which store the information about the position and size of each robot on the image are save in *JSON* format. An exemplary annotation for image 3.6 b) would look like the following.

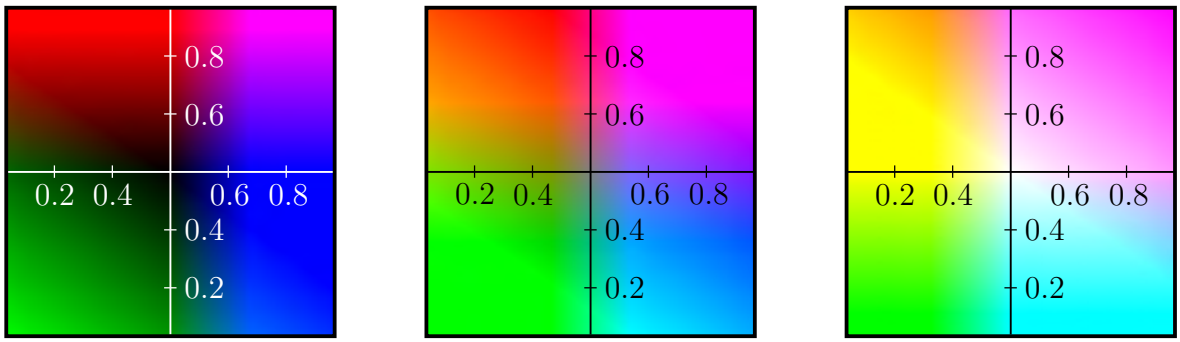
```
{″objects″ : [
    {″robot″ : [
        [″0.1969″,″ 0.4125″,″ 0.0734″,″ 0.1917″],
        [″0.1703″,″ 0.4125″,″ 0.0344″,″ 0.1104″],
        [″0.2609″,″ 0.4146″,″ 0.0219″,″ 0.1042″],
        [″0.3844″,″ 0.4062″,″ 0.1016″,″ 0.2000″],
        [″0.5969″,″ 0.4042″,″ 0.0375″,″ 0.1208″],
        [″0.8531″,″ 0.3917″,″ 0.1047″,″ 0.2354″]
    ]}
]}
```

The regions of interest for robots are given in normalized values with respect to the image width and height. Each instance is defined by four values. The first two values denote the position of the upper left corner of the region of interest. The first entry is the horizontal and the second entry the vertical position. The last two values denote the width and height of the region of interest. The third value defines the width and the fourth value the height. A benefit of this nested structure is, that it can be easily extended by further object types like balls or goal post locations.

In the further work only the abbreviations presented in Table 2 will be used to denote the respective sets.

3.3.1 Color Model

The native color model of the Nao is YUV422 [26]. It splits colors into a luminance component Y and two chrominance components U and V. Figure 3.7 illustrates the UV plane with three different Y values.



a) UV plane, $Y = 0$. b) UV plane, $Y = 0.5$. c) UV plane, $Y = 1$.

Fig. 3.7: UV plane of YUV color model with three different Y values.

Typically the values of U and V range from zero to one with white, grey and black at $U = V = 0.5$.

However, this work uses images given in the red, green and blue (RGB) color model because this improves the analysis of convolutional filters in the first layer. Unlike YUV, in RGB images a pixel color is defined by the ratio of red, green and blue. Only the brightness is defined by the absolute values (s. Figure 3.8).

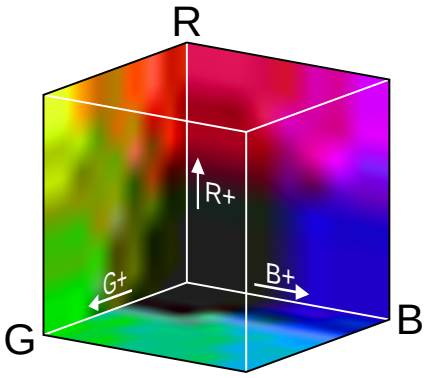


Fig. 3.8: Representation of the RGB color model as cube [27].

Typical convolutional filters of DCNNs have the same depth as their input. The filter weights are not limited to the input space but can be normalized with respect to the value range of the input space. Since the ratio in the three depth channels of the filter is not effected by this operation, each filter can then be plotted as RGB image. The resulting color shows, which color channels and configuration influences the output of these filters the most.

3.4 Training Methodology Contrast Based Object Localization

As the structure of the implementation is not altered and has a convex and more stable learning behavior compared to a DCNN, the description and evaluation of learning will not be as detailed as for the DCNN in this work. To prevent overfitting the SVMs receives the same augmented data as training and test set and will be evaluated using the evaluation metrics of section 1.4. However, the effective training set size differs, because the implementation trains with randomly sampled background crops as negative examples and discards empty background images. Furthermore, L^1 regularization is applied. The difference to weight decay or rather L^2 regularization in this context is the 1-norm instead of the 2-norm. L^1 regularization does not favor small dense weights but only a small sum of weights. This again can lead to sparse weights, which favor the distinct generalized objectness model depicted in Figure 2.3.

3.5 Training Methodology Deep Convolutional Neural Network

The training will be performed in end-to-end fashion. Only inputs and ground truth output are provided during training. To prevent overfitting data augmentation, weight decay and periodical validation are performed. The standard gradient descent requires learning rate adjustments, when it is getting close to local optima and converges slowly compared to modern improved variations. Therefore, an enhanced gradient descent method is used. It performs naturally performs a form of step size annealing [28] and is described below.

3.5.1 Optimization

Optimization is performed with a more sophisticated version of gradient descent called *Adam*, whose name is derived from adaptive moment estimation. The content of the following is therefore based on [28]. Let t denote the training step and W all trainable weights. $J(W)$ is the cost function to be optimized with respect to W, and g_t the respective gradient at training step t. Furthermore m and v are moment vectors, which have the same dimension as the weights W and accumulate the weight gradients of previous training steps. α is again the learning rate, ϵ an offset to prevent division by zero and β_1, $\beta_2 \in [0, 1)$ are decay factors for the accumulated weight gradients m and v. g_t^2 denotes the elementwise square $g_t \odot g_t$ and β_1^t, β_2^t the respective βs to the power of t. m and v are initialized with $m_0 = 0$ and $v_0 = 0$. The suggested hyperparameter values are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The weight update rule becomes the following:

$$g_t = \nabla J_t(W_{t-1}) \tag{3.2}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.3}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.4}$$

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.5}$$

$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3.6}$$

$$W_t = W_{t-1} - \alpha \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \tag{3.7}$$

After computing the gradient in 3.2 the moment vectors are calculated, such that the moments consist of a ratio of the current gradient and the accumulated gradients. The ratio is defined by the decay rates β_1 and β_2. This counters fluctuations close to a minimum, if the learning rate and thus the weight updates are too large. The gradients are squared in 3.4 because they are used for scaling of m_t in 3.7. $\widehat{m}_t$ and $\widehat{v}_t$ are further scaled moment vectors, such that the quotient of 3.7 damps the accumulated gradients m_t in the beginning to prevent divergence short after initialization. The damping decreases as t increases until $\widehat{m}_t \approx m_t$ and $\widehat{v}_t \approx v_t$. Finally both moment vectors are divided elementwise. The square root has to be computed because the gradients are squared in 3.4 to ensure positive gradients in 3.7. ϵ has to be added as safety measure to prevent division by zero. Note that this holds when $\beta_1 < \beta_2$, otherwise the cost function might diverge soon after initialization or later because m_t does not decay faster than v_t.

3.6 Summary

This chapter discussed typical issues, which arise during training of machine learning methods and presented approaches to resolve these. Furthermore the data base was introduced with a detailed view of the image origins and their color model. Lastly the training methodologies for both of the tested approaches were presented. The presented techniques and information are used to train and evaluated both object localization approaches, which is discussed in the following chapter.

4 Evaluation

This chapter evaluates the proposed methods of chapter 2 in terms of precision, recall and computational time. Since especially DCNNs face more difficulties during training due to large numbers of parameters and non-convexity of their solution compared to SVMs, the evaluation emphasizes the DCNN learning behavior. Nevertheless, the contrast based approach is evaluated with regard to the default implementation by the author of [16] and model or rather configuration selection for the final test. In reference to chapter 3 the augmentation techniques are explored in their effect to improve the result and prevent overfitting in DCNNs. Based on error progression and filter analysis a suitable DCNN structure is selected.

4.1 Evaluation Setup

The base data set of 1748 images is divided into $\frac{3}{5}$ for training, $\frac{1}{5}$ for validation and $\frac{1}{5}$ for testing. Both machine learning methods train with the training set and use the validation set during training to avoid overfitting. To prevent the training and validation process to overfit on the validation set, the test set is used for the final test of both object localization approaches.

The training and evaluation will not be performed on the Nao robot due to the limited time frame of this work. Thus, it is performed on a desktop computer with Linux Mint 17.2 Cinnamon 64-bit OS, an AMD Phenom(tm) II X4 965 processor and 8 GB RAM. No multi-core or GPU computation is used during evaluation since the Nao does neither have multiple cores nor a built in GPU. The contrast based object localization approach is implemented in *C++* and publicly available by the author of [16]. The implementation of the proposed network structure is performed in *Python* with *TensorFlow*, *NumPy* and *PIL*. Computation time is measured on the test machine using the standard *time* library in Python and *sys/time.h* in *C++*. The respective measurements can only present a relational comparison between both approaches and do not reflect the actually necessary processing time on the Nao. However, if the duration of process exceed the predefined limit of 30 ms, it will also be too slow on the weaker processor of the Nao.

4.2 Contrast Based Object Localization Using BING

Because the structure of this approach is not altered compared to the DCNN, the publicly available *C++* implementation of the respective author is used [16]. After applying data augmentation in form of horizontal flipping and image section cropping, the test set contains 61,430 images. No modification of the brightness is applied because it is also not applied to the training images of the DCNN, which is explained more detailed in 4.3.3. BING automatically generates background crops as negative learning example and discards empty background images. This changes the augmented training set size from 61,430 to 26,834. It offers the option to vary the quantization base q, the number of proposed regions of interest p_{RoI} in each quantized window and a NMS value to reduce the

number of overlapping proposals. Increasing q decreases the number quantized windows (s. Figure 2.2) and ultimately the absolute number of proposed regions of interest.

4.2.1 Model Selection

The default configuration of this approach resizes the original image to 36 different sizes (including the original) and predicts 130 bounding boxes per size. The two SVMs rank each bounding box and denote how likely it contains an object. However, 1,000 proposals are too many for a Nao robot to process given the limited cycle time of 30 ms.

Assuming that a maximum number of 10 robots are on the field, the maximum number of detectable robot objects by the viewing robot is 9. However, this unlikely case only occurs, when all robots are in the visible area and have the same distance to the viewing robot, so that all fall into the same quantized window size for detection. Thus, 9 proposals per quantized window are sufficient and the maximum number of proposal per quantized size is reduced to $p_{\text{RoI}} = 9$. An additional reduction of proposals and computation time can be achieved, by increasing the quantization base to $q = 4$. This means, that the image width and height is divided by 4 after each quantization step.

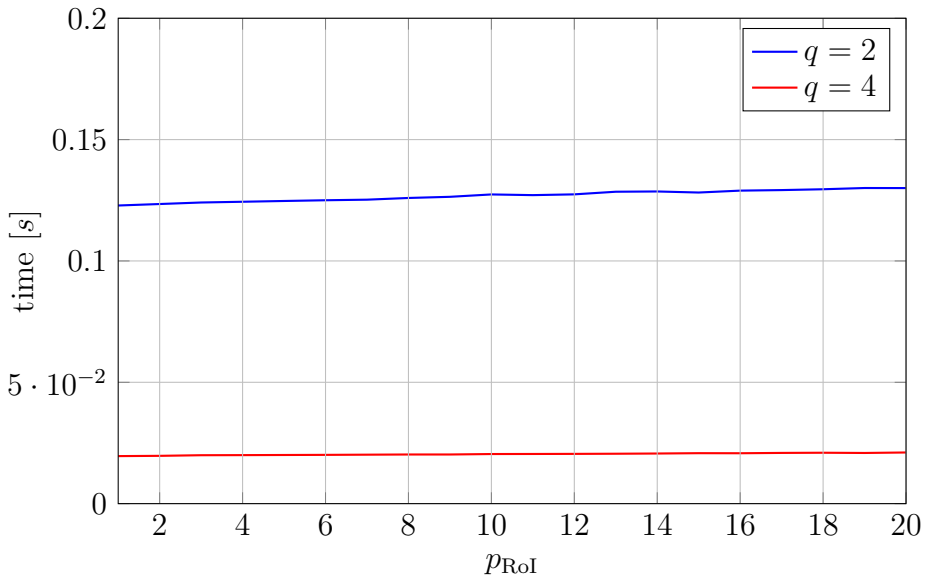


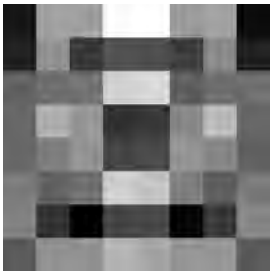
Fig. 4.1: Average prediction time for one image for $q = 2$ (blue) and $q = 4$ (red) as p_{RoI} increases from 1 to 20.

4.1 demonstrates that the quantization size has the largest impact on the computation time. In fact, the average prediction time for one image for $q = 2$ is always bigger than 120 ms, which does not lie within the mentioned constraint of 30 ms. Since the original aim is to predict only a small number of high quality proposals, a configuration of $q = 4$ and $p_{\text{RoI}} = 9$ is chosen.

The performance on the test set will be discussed in section 4.4.

4.2.2 Model Evaluation

One full training takes approximately 15 minutes due to the large number of training images. The learned model of a robot object below does not look as generalized and distinct as the learned model of a general object, which is depicted in 2.3.



a) Learned Nao robot model **w**.

b) Exemplary image of Nao.

Fig. 4.2: Learned Nao robot model **w** a) and an exemplary image of a Nao to demonstrate the similarities.

This can be traced back to the fact, that the original set of training data only contains marked robot instances. This yields a more specialized model of an object. The gradient change in the middle results from the jersey which most Naos wear on the images. Above and below the jersey are always white robot parts (i.e. head and legs) which explains the high gradient right above and below the dark spot in the middle. The gradient is not as salient on the sides because there is often a gap between the robot arms and its torso. These gaps often display different parts of the background and can therefore lead to a lower average gradient. Performing object localization on a validation image yields the following result.

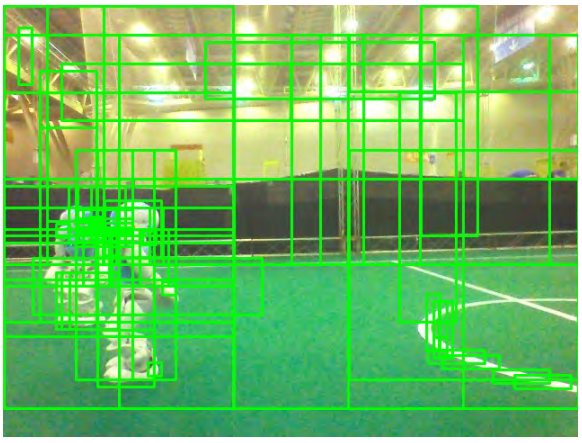


Fig. 4.3: Demonstration of all predicted boxes for a validation image using BING with $q = 4$ and $p_{\text{RoI}} = 9$.

The density of bounding boxes increases around the robots, which is a good sign of the

system's recall ability. However, the number of predictions is still too high, although the number of proposals per quantized window size was already lowered from 130 to 9. The system reaches a precision of 0.4 % and a recall of 18.5 %. Increasing the NMS value from 2 to 5 worsens the system performance to 0.4 % precision and 14.6 % recall. To simply reduce the number of proposals only the highest scoring predictions shall be accepted. Assuming again a maximum number of 9 robots at a time leads to the following.

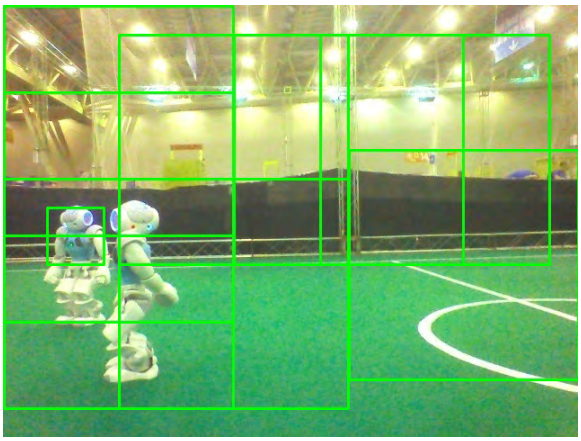
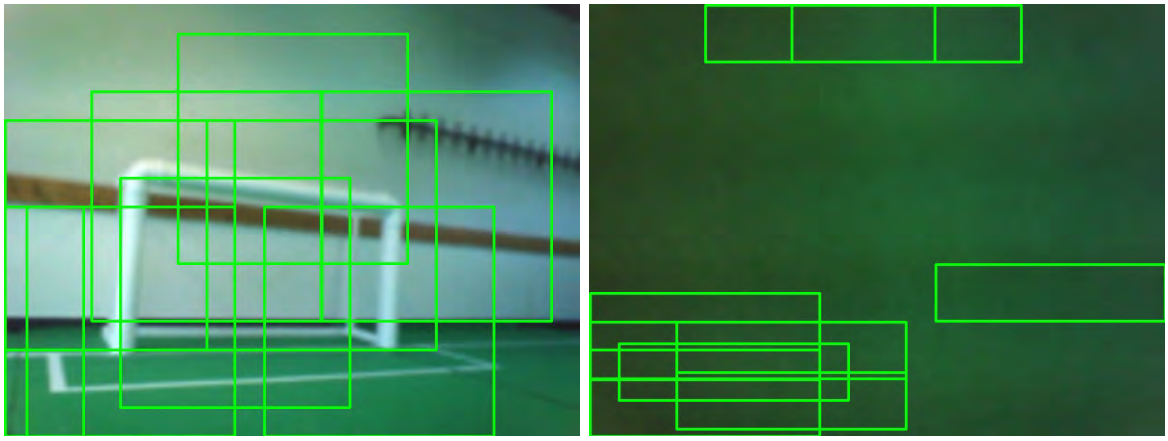


Fig. 4.4: Demonstration of top nine highest ranking predictions.

The precision increased to 1.8 %, whereas the recall decreased to 11.1 %. Another interesting observation can be made, looking at the predicted bounding box sizes. When only the highest scoring predictions are accepted, the average size of the bounding boxes is larger than before. The reason for this is based on the training set. Most images with robot instances depict Naos from a close range since they are more common and in terms of in-game situations also more important. To enlarge the training set, two augmentation methods were applied. Horizontal flipping does not alter the scale, so that it does not introduce a significant change in the window size scoring. The image section cropping augmentation on the other hand does scale the resulting image because the cropped area is resized to its original dimensions. Since the cropping is performed in a sliding window fashion, the scale is constant and therefore biases the SVM towards a larger scale than reflected by the original images. A random section cropping, which only enforces a constant aspect ratio, would hence be a more robust augmentation method.

The previous proposal images showed that as long as robots are present, the major part of the predicted boxes are centered in this area. The figures below show the results for images, which do not contain a robot.



a) Typical scene without a robot instance. b) Prepared empty scene.

Fig. 4.5: Bounding box predictions of BING for two validation images, which do not contain a robot instance to test for false positives. a) could occur in a normal game, whereas b) only shows a small part of the soccer field.

The number of false positives is similar to the general number of predictions. Considering that the predictions are centered around robots, if present, leads to the conclusion that the rating of the generated proposals is not sufficient for the task of highly selective object localization. Too many background patches exhibit BING features similar to the learned model. This weak selectiveness results from the original intention of this approach to be a general object proposal generator, which mainly focuses on high recall rates, with a subsequent classifier.

In order to be used in SPL robot soccer, the BING approach could be modified, so that it needs less computational time and only proposes very few qualitative regions of interest, i.e. high precision. BING needs 20 ms for one image on the test machine but will need more time on the weaker processor of the Nao. Figure 4.1 demonstrates that the processing time mainly depends on the quantization size q and therefore on the number of performed window quantizations. Thus, reducing the input dimensions of the image, has two benefits. Firstly, in the scenario of SPL robot soccer there are no such tiny foreground objects, which could be detected by an 8×8 pixel window on 640×480 input. These proposals lead to many false positives and lower the precision. If the ball should be detected, the halved input size would be sufficient. This would make a ball with a size of 16×16 on a 640×480 image detectable. If only robots are considered, the original input dimensions could be further reduced because robots are larger objects. Robot localization is mainly used for obstacle avoidance and path planning, which only makes robots in close distance important. Assuming that the smallest robot instance, which should still be detectable, has a height of approximately 100 pixels on a 640×480 input, it could be detected by an 8×8 pixel window after scaling it down with a factor of roughly $\frac{1}{12}$. Secondly, a smaller input size means less quantization operations and therefore less computational effort. A higher selectiveness could also be achieved by increasing the dimension of the sliding windows to learn a more detailed model. This however, would decrease the computational efficiency

and require further research to determine the optimal size for capturing sufficiently enough details, whilst being efficient.

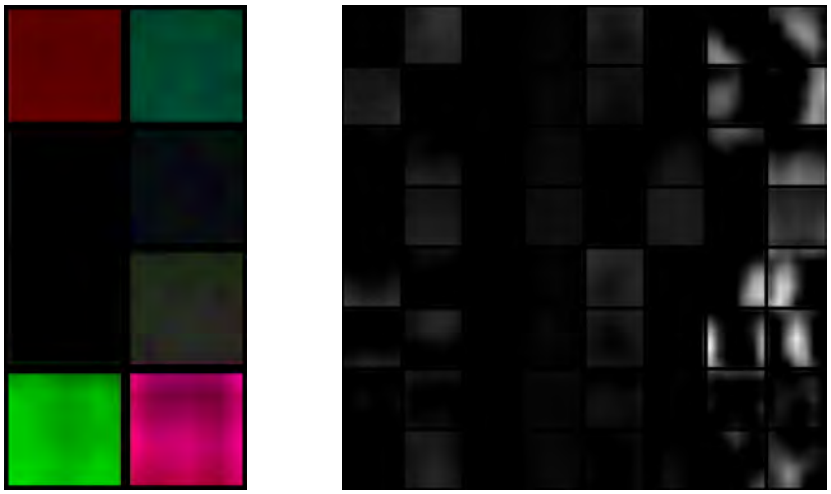
Since the general idea and efficiency of BING features is interesting and has the potential to be used on a real time system, further research could be conducted with mentioned modifications.

4.3 Deep Convolutional Neural Network Based Object Localization

Training is performed with batches of training data with a size of 32. For the evaluation of regularization and data augmentation only one proposed network structure of 1 is used. The configuration of 4 filters in the first and 8 filters in the second layer with a filter size of 7×7 is chosen for the tests. The choice is reasonable because it has, compared to the other proposed configurations, the most parameters (5,453) and is therefore a candidate for overfitting, if trained on only the base training set.

4.3.1 Convolutional Filter Interpretation

Before evaluating the proposed methods a short introduction into the upcoming filter plots should be given. The used DCNN always has two layers and thus two different sets of filters.



a) Filters of first layer.

b) Filters of second layer.

Fig. 4.6: Plot of first and second layer filters. Each row represents one filter with pairwise positive and negative parts. The first layer filters can be interpreted as RGB image.

Figure 4.6 a) depicts a set of first layer filters sorted from top to bottom (i.e. first row $\widehat{=}$ first filter, second row $\widehat{=}$ second filter etc.). The first layer filters will directly operate

on the input RGB image and therefore have a depth of three each. They are plotted and interpreted as RGB images which makes the analysis of extracted features easier than examining each depth channel individually. Because the filters are not bound to positive values they can also learn negative values. Therefore the plot contains the positive filter values on the left side and the negative ones on the right side. All values are normalized by the largest absolute value of the filter set in each layer to give an impression of the most influencing filter.

Figure 4.6 b) depicts a set of second layer filters. They operate on the filter maps created by the previous convolutional layer and can therefore not be interpreted as RGB image. One row still represents one filter, whereas the depths may vary depending on the number of filters in the previous layer. Since the filter cannot be interpreted as RGB image anymore, each depth layer of a filter has to be plotted separately. Additionally each depth layer has to be split into negative and positive parts again, so that one depth layer is always represented as pair of columns.

4.3.2 Weight Decay

To test the effect of weight decay, the network is only trained on the training set of the original data base. For validation, the respective validation set is used. The system is bound to overfit with 5,453 parameters and just 1,048 training images.

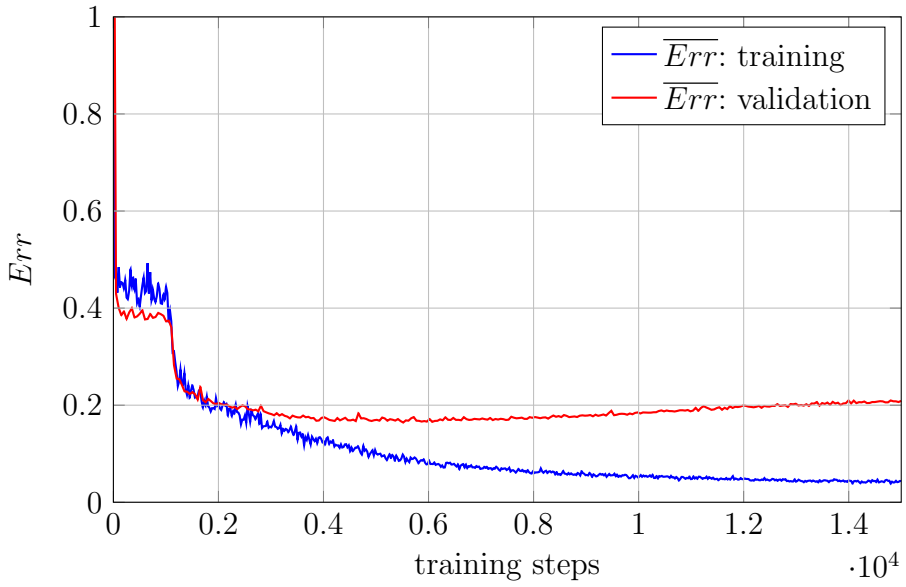


Fig. 4.7: Progress of network error on training (blue) and validation set (red) during training without usage of weight decay or data augmentation.

Figure 4.7 shows a plot of training and validation error progression during training. The training error clearly converges to zero, whereas the validation error slightly diverges over time. The validation error reaches its minimum at approximately training step 4,500.

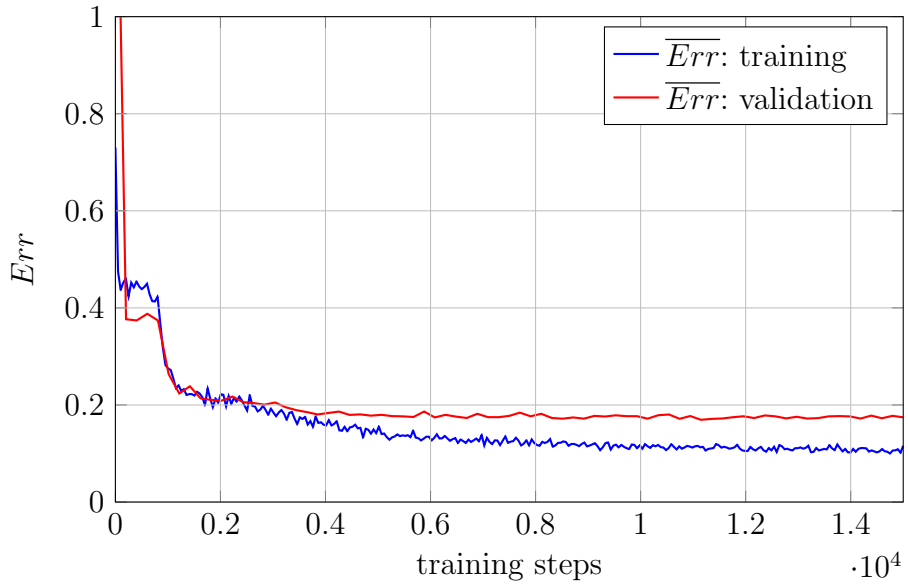
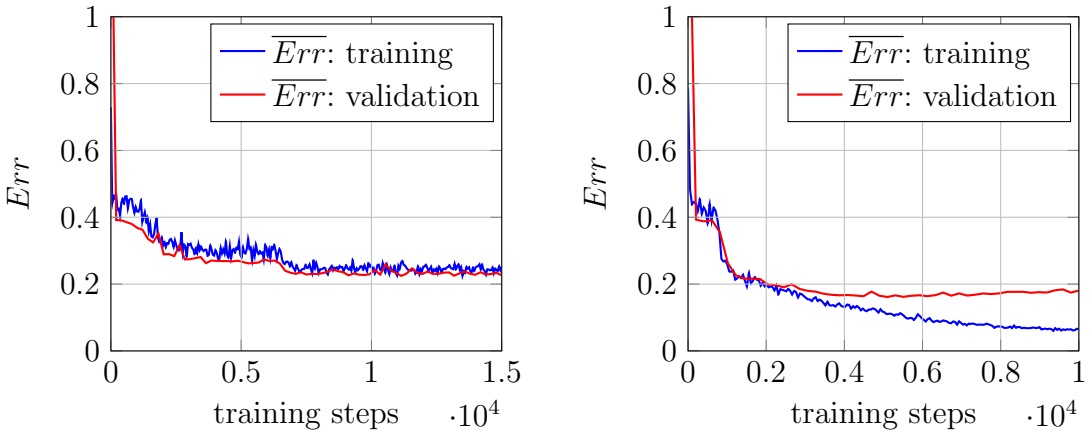


Fig. 4.8: Progress of network error on training (blue) and validation set (red) during training with usage of weight decay regularization.

Figure 4.8 shows the same progression with activated weight decay. The regularization penalty (s. Figure A.1) is not included in the error plot because this would introduce an additional offset to both errors and make the comparison more difficult. It is clearly visible, that the training error does not converge to zero as fast over time. The regularization weight is set to $\lambda_{reg} = 0.01$ and was determined by evaluating the learning progress with $\lambda_{reg} = 0.001$, $\lambda_{reg} = 0.01$ and $\lambda_{reg} = 0.1$.



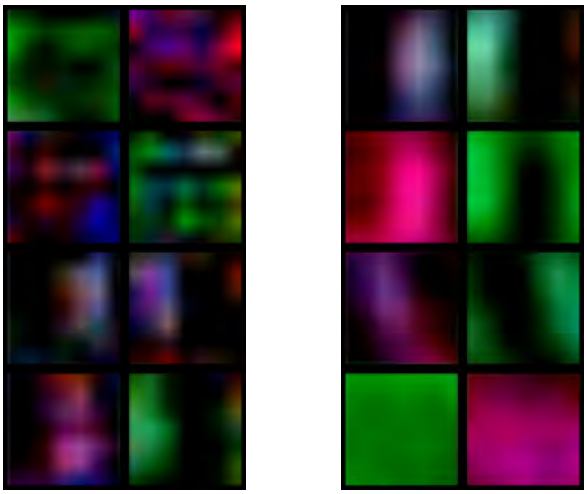
a) $\lambda_{reg} = 0.1$.

b) $\lambda_{reg} = 0.001$.

Fig. 4.9: Training with different regularization weights λ_{reg}.

$\lambda_{reg} = 0.1$ penalizes the system too heavily, so that it is not able to learn diverse features

and underfits (s. Figure 4.9 a)). $\lambda_{reg} = 0.001$ imposes to weak constraints, resulting in similar error progression to the system without regularization (s. Figure 4.9 b)).



a) Filters learned without weight decay.

b) Filters learned with weight decay.

Fig. 4.10: Plot of first layer filters after training without a) and with b) weight decay ($\lambda_{reg} = 0.01$).

Figure 4.10 depicts the first layer filters of both setups. 4.10 a) shows very specialized characteristics, i.e. single peaks in different color channels. In contrast 4.10 b) shows characteristics, which could be interpreted as extractable features of an image, like red and green color, or a vertical edge. This leads to the conclusion that weight decay induces more robust filter learning because it penalizes sparse weights more than evenly distributed weight values.

4.3.3 Data Augmentation

The training and validation set partitioning has to be changed to evaluate the effect of scaling HSV color model channels. If random images would be selected for validation, the distribution of bright and dark images would be similar to the training set. To enforce a difference in lighting conditions the validation set should be from another location unrelated to the training data. Thus, all sets but the China set are used for training. The China set is selected in particular for validation because it has relatively bright images and differs from the rest in that regard.

Training	H	S	V
Brazil	0.436	0.403	0.366
RoHOW	0.308	0.577	0.424
TUHH N	0.404	0.480	0.520
TUHH Lab	0.254	0.441	0.547
HTWK Lab	0.336	0.494	0.522
$E(H_{\text{train}})$, $E(S_{\text{train}})$, $E(V_{\text{train}})$	0.364	0.495	0.445

Tbl. 3: Averaged HSV values of individual training sets and respective expected values.

Validation	H	S	V
China	0.320	0.530	0.625
$E(H_{\text{val}})$, $E(S_{\text{val}})$, $E(V_{\text{val}})$	0.320	0.530	0.625

Tbl. 4: Averaged HSV values of individual validation set and respective expected values.

The tables above contain the averaged HSV values of each set and the respective expected values. Comparing the expected values yields, that the sets only differ by very small hue and saturation values (0.044 and 0.034) but have a relatively big difference of 0.18 in the expected brightness value. To test the effect of scaling the brightness value of images to artificially introduce different lighting conditions the training set is modified. The V channel of each image is scaled up by 1.4, which results in a new expected brightness value of 0.623.

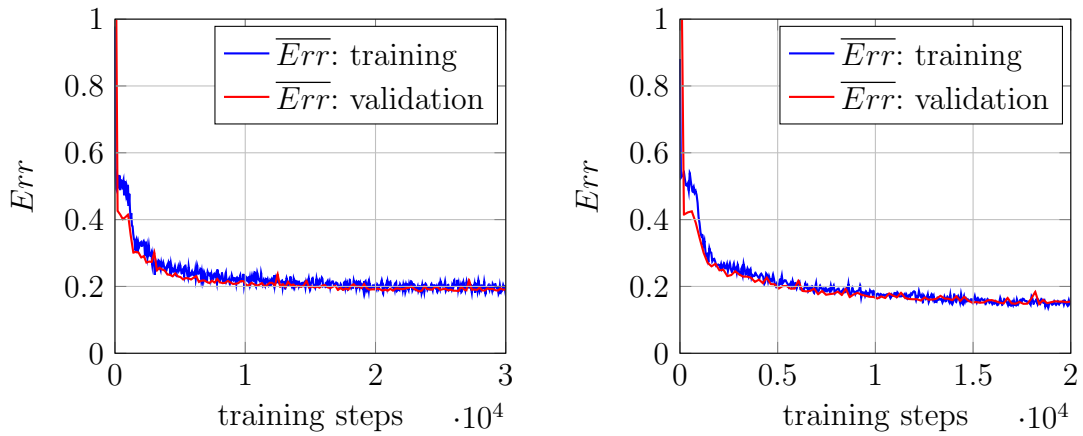
To evaluate and compare the performance on both sets the lowest validation error value is taken. Since this value may vary depending on the random weight initialization both sets were evaluated three times (s. appendix for error progression plots A.1.2). The average of the lowest validation error values yields $\overline{Err} = 0.366$ and $\overline{Err}_{\text{augmented}} = 0.363$. The average error value was decreased by only 0.003, which shows, that scaling brightness values has almost no effect on the validation error, even if there is a known average brightness difference and the scaling is targeted at this difference. Randomly scaling brightness can therefore not introduce robustness against changing lighting conditions in this localization network.

4.3.4 Model Selection

To determine, which filter size and number are best suited for the localization, all structures are trained until there is no more convergence of the validation error to zero or it diverges. Section 4.3.2 already showed, that applying weight decay during training does not completely prevent overfitting but nevertheless leads to less complex and thereby more robust features, even if the training set size is small (1048) compared to the trainable parameter number (5453). To further enhance the robustness, the training and validation is performed on original and augmented data to introduce additional deviations of robot

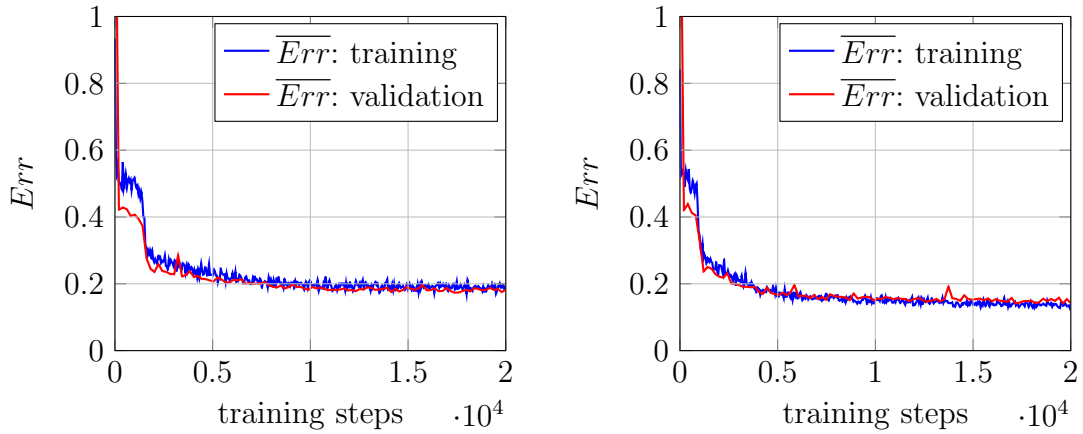
positions. The training set contains the original images, the respective horizontally flipped versions and cropped parts with a cropping window size of 480 × 360. This results in 61,430 training images. Periodical validation during training should only take a small portion of the actual training duration. Thus, only horizontal flipping is applied. This results in a validation set of 700 images. With a validation interval of 100 training steps and a training batch size of 32, the smaller structures, which use two filters in the first layer and four in the second, take approximately 45 minutes for 10,000 training steps. The larger ones need roughly 55 minutes for 10,000 training steps.

The notation of each structure is abbreviated to $w_{\text{filt}} \times h_{\text{filt}}$ - $NoF_{\text{layer 1}}$ - $NoF_{\text{layer 2}}$, where the depth of each filter is always equal to its input. E.g. 5×5 - 2 - 4 can be interpreted as a network structure with two 5×5 filters in the first layer and four filters in the second layer.



a) 5×5 - 2 - 4.

b) 5×5 - 4 - 8.

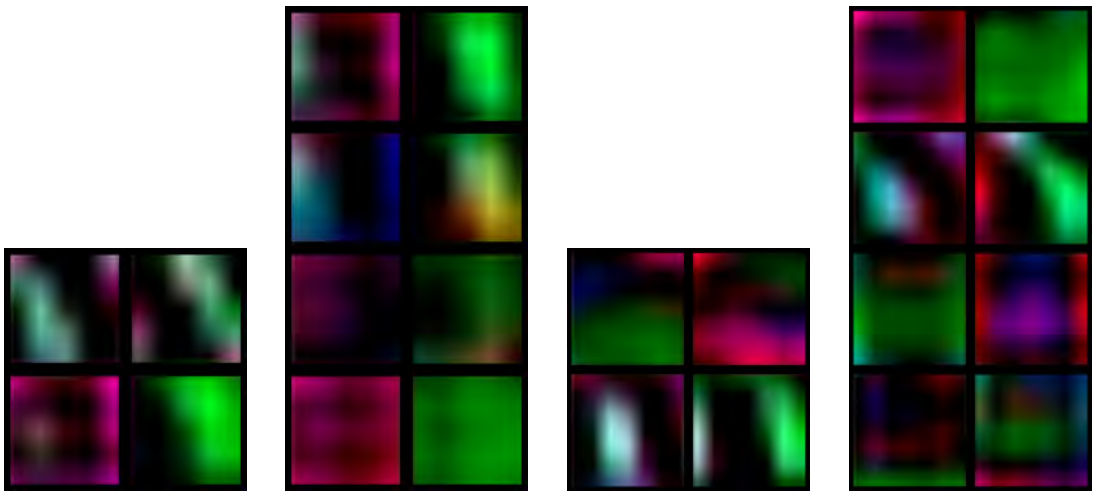


c) 7×7 - 2 - 4.

d) 7×7 - 4 - 8.

Fig. 4.11: Error progression of each denoted structure during training without weight decay.

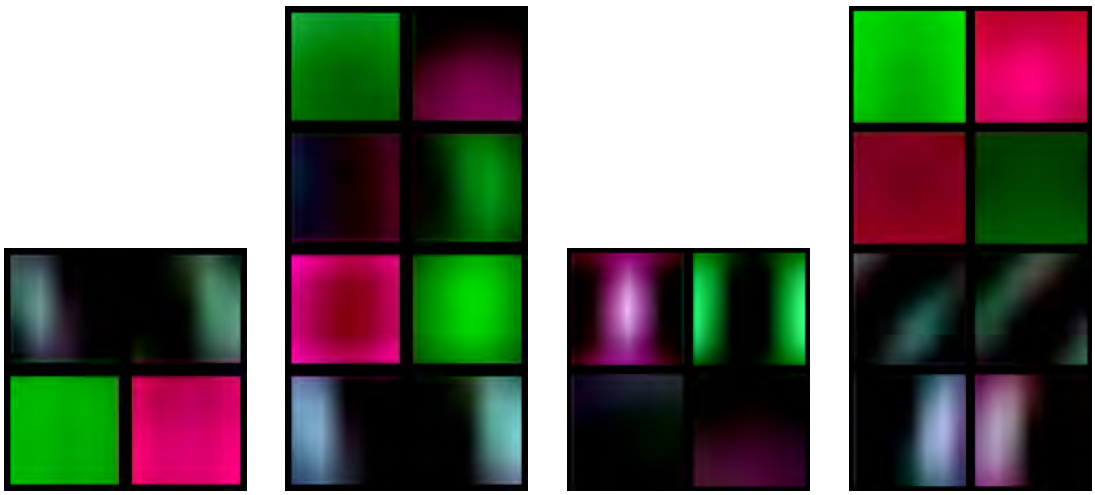
The plots above show the error progressions of the four proposed network structures without utilizing weight decay during training. Regarding the validation error, none of them shows signs of overfitting as training progresses. Plotting the learned filter however demonstrates, that the networks still employ a slight overfitting behavior. The filter maps of Figure 4.12 still have a patchy weight distribution over the different color channels.



a) 5×5 - 2 - 4. b) 5×5 - 4 - 8. c) 7×7 - 2 - 4. d) 7×7 - 4 - 8.

Fig. 4.12: Plot of first layer filters for each network structure.

Nevertheless some filters show the tendency to edge like features e.g. 4.12 a) first row, 4.12 b) first and second row. To further improve the learned filters, weight decay with the previously tuned parameter $\lambda_{reg} = 0.01$ is applied.



a) 5×5 - 2 - 4. b) 5×5 - 4 - 8. c) 7×7 - 2 - 4. d) 7×7 - 4 - 8.

Fig. 4.13: Plot of first layer filters for each network structure.

Figure 4.13 shows the positive effect of weight decay. All filters have a more balanced

weight distribution and are therefore less specialized on the training set. The edge like feature filters are more distinct and less color specific, e.g. 4.13 b) row 4, 4.13 c) row 1. The filters of the second layer show similar characteristics but are only depicted in the appendix because they are more difficult to interpret (s. Figure A.3 and A.4). They are more in numbers and extract features of features and can therefore not be interpreted as RGB image anymore. Taking a look at the measurements of validation error, recall and precision, supports the stated interpretation.

Network Structure	**Top** Err_{val}	**Recall**	**Precision**	**Network Evaluation**	**Complete Evaluation**
5×5 - 2 - 4	0.185	0.263	0.536	5 ms	33 ms
5×5 - 4 - 8	0.147	0.317	0.606	5 ms	33 ms
7×7 - 2 - 4	0.174	0.260	0.558	8 ms	37 ms
7×7 - 4 - 8	0.141	0.321	0.602	8 ms	37 ms

Tbl. 5: Measured results after evaluation on validation set after training without weight decay.

The performance of the network structures with more filters is clearly superior in validation error, recall and precision. The localization performance of the weight decay regularized model yields similar results. Comparing the results directly points out that the overall recall and precision decreased for the 5×5 - 2 - 4 and 7×7 - 2 - 4 networks. This can arise due to the fact that two filters might not be enough to extract all the necessary features of the input data. Therefore, a small overfitted system can outperform a more generalized system on the training and validation set.

Network Structure	**Top** Err_{val}	**Recall**	**Precision**	**Network Evaluation**	**Complete Evaluation**
5×5 - 2 - 4	0.193	0.238	0.510	5 ms	33 ms
5×5 - 4 - 8	0.179	0.242	0.558	5 ms	33 ms
7×7 - 2 - 4	0.191	0.225	0.515	8 ms	37 ms
7×7 - 4 - 8	0.164	0.292	0.572	8 ms	37 ms

Tbl. 6: Measured results after evaluation on validation set after training with weight decay ($\lambda_{reg} = 0.01$). Because weight decay regularization generally yields better filters for each structure, only the regularized structures were trained three times (s. A.5, A.6, A.7) and evaluated with respect to their average performance.

The performance of the larger structures 5×5 - 4 - 8 and 7×7 - 4 - 8 displays similar results. The overall performance is worse than the performance of the respective non regularized structures but the filters show a better generalization ability. The regularized networks are therefore preferred over the non-regularized ones. The network structures with 5×5 filters need 33 ms for one complete evaluation, from loading the image into a resized array, feeding it into the DCNN and interpreting the output. The actual network evaluation takes only 5 ms. The networks with 7×7 filters need 8 ms for one network eva-

luation and 37 ms for a complete evaluation. With a minimum complete evaluation time of 30 ms all proposed network structures will fail on the Nao using this implementation setup. However, there are two interesting observations.

1. The time for one network evaluation does not increase, although the number of filters is doubled, whereas the filter size directly effects the computation time. It is possible, that this is caused by internal processes of the *TensorFlow* library because the number of necessary floating point operations of the 5×5 - 4 - 8 and 7×7 - 4 - 8 are closer to each other than the corresponding structures with half the filters (s. Table 1). A faster performance could therefore be achieved by utilizing lower level libraries for small DCNNs, at least for the actual feed forward applications.

2. The major part of computational time and effort is consumed not by the network itself but by the data preparation. Since the image data can be accessed directly on the Nao and efficient downsampling via scanlines is already performed, a Nao adjusted implementation would have a better computational performance.

If the resulting evaluation times of Table 5 and 6 are independent of the implementation and used libraries none of the proposed models would run within the predefined time frame of 30 ms. However, the calculated number of necessary floating point operations and the above stated observations suggest, that these network structures are suitable for a real time application on the Nao. Because the 7×7 - 4 - 8 has demonstrated the best overall localization ability, it is selected for the evaluation on the test set.

The performance on the test set will be discussed in section 4.4.

4.3.5 Model and Filter Evaluation

This section provides an analysis of the internal network output based on selected validation images. This allows a better understanding of each filter because the actual produced output is visualized. The previously selected network structure (7×7 - 4 - 8) is used for this purpose.

Fig. 4.14: Demonstration of separate first layer filters and *max-pooling* outputs for a given input image.

Figure 4.14 shows the inter process output of the first convolutional layer and subsequent *max-pooling* layer for an example input image of the validation set. Note that the filter values and output layer maps are normalized with respect to the largest filter value in a filter or with respect to the largest output value in a filter map. Thus, the influence of each filter and output is visualized by its color intensity. The intensity rises from black and dark colors to bright and white. The first filter scores green color positively and red color negatively. This leads to an extraction of the soccer field in the general case. White areas, which consist of red, green and blue values to the same amount, are scored with zero since the filter values for the red and green channel cancel each other out. The second filter is an inverted version of the first, although the ratio between the red and green channel is different. The red filter channel is stronger and therefore highlights red and bright white spots. Filter three shows a color insensitive behavior. It reacts with a high output on inclined edges but is not distinctive enough. The resulting filter map is mainly grey with only a few slight accents of oriented edges. A more distinctive filter for edges should not react with positive outputs on a plain soccer field. The downsampling of *max-pooling* worsens this indistinctive characteristic and makes this map very weak in terms of information content. This can be the result of the limited filter number of four. To draw important information of inclined edges, more filters with different orientations and stronger distinction ability would be necessary. The fourth filter shows a better example of a color insensitive edge filter. Although the positive part has a blue and the negative part a red color tint, the filter highlights vertical edges in the resulting filter map and discards plainly colored areas, like the actual soccer field.

The filter maps actually consist of positive and negative values but the latter ones are neglected as they are very small and therefore not visible on the output maps. This results from the leaky-ReLU with a scaling factor of 0.1 in negative value range. The complete output can be seen in the appendix A.8.

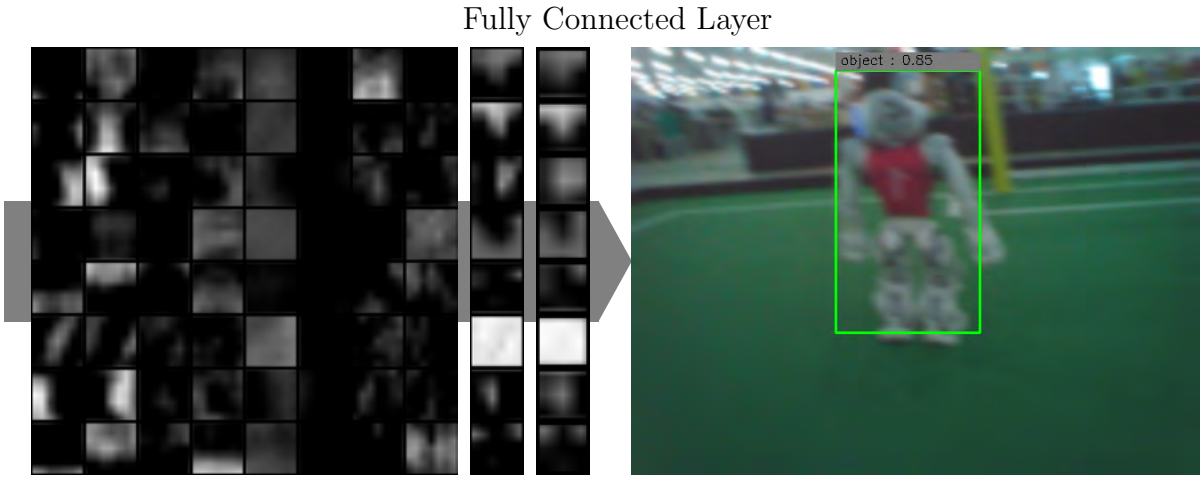


Fig. 4.15: Demonstration of separate second layer filter and *max-pooling* outputs and the final interpretation of the DCNN.

Figure 4.15 shows the second stage of the internal process. The rightmost output of Figure 4.14 is the input for the second convolutional layer which is depicted above. The filters are sorted from top to bottom and the depths layers from left to right. Each filter's depth layer appears as pair and is again separated in a positive part (left) and negative part (right). E.g. the first two columns are the depth layers of all eight filters, which convolve the first filter map of Figure 4.14. An interesting general observation of the filters is that the negative weights have a larger influence in this layer except for the fifth and sixth columns, which are the reason for this behavior. Because the third filter of the first layer produces a filter map with only little distinctive information, these parts can hardly be further interpreted resulting in a general positive offset in most of the filters. The filters operating on the first two filter maps of layer one now "look" for edge and shape like structures. The last two columns of the second convolutional filter plot show the filter layers, which operate on the map of vertical edges. The filters do not show a very generalized human interpretable characteristic and do not have as much influence on the final result as the filters operating on the first two outputs, which can be seen by the intensity of the grey value. The first two output filter maps show a more downsampled version of an inverted soccer field extraction, whereas the fourth output shows the extracted soccer field. The third and seventh output map show the positions of the largest vertical edges on the image which is basically the position of the Nao in the middle. The vertical edges could be extracted that well because most of the Nao surrounding background is constantly green. Output five and eight are the background parts of the image, which are not soccer field and not Nao. The sixth output does not show any distinctive information and rather seems like a constant. The responsible filter shows a similar feature characteristic of the first layer. It depicts an inclined edge, which again results in a too even and indistinct output.

Further insight can be drawn from another example of detected false positives. The figure

below shows only the two filter maps after convolution (without *max-pooling* to provide a more detailed view) and the resulting bounding boxes.

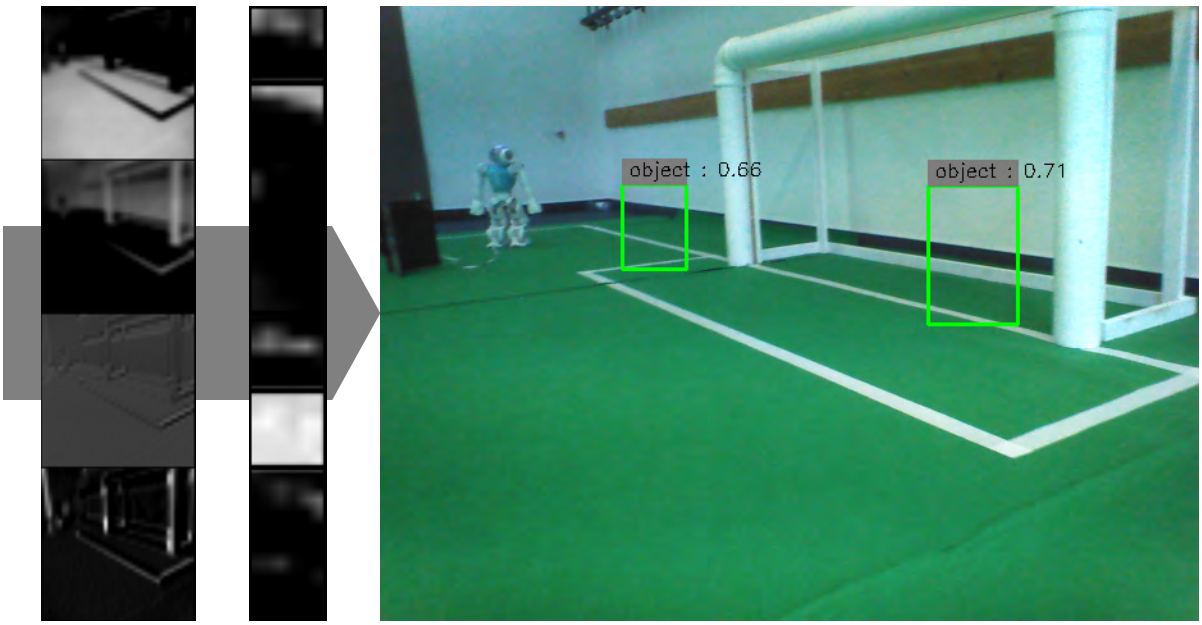


Fig. 4.16: Illustration of first and second convolutional layer output and the resulting predictions.

The first convolutional layer output performs again field and background extraction. However, the Nao in the top left corner is too small and does not have enough contrast to the background, so that it is basically treated as background. It also shows a bias towards red color of the network filters because in the overall image set, which includes Naos with blue, red and without jerseys, red is the most distinctive one. The most salient vertical edges are the goal posts and the door in the top left corner. The outputs show the same behavior as described for the first example image. However, two false positives are predicted. Looking at output map three and seven, shows the computed position of the most salient vertical edges. The intensity of the ones in output seven is larger and also more likely for a Nao because its located in the centric region of the image. Therefore two false bounding boxes are predicted. The "false" location, which should actually be on the goal posts, can be traced back to the strong downsampling to a 3×3 grid.

These results show that the predictions are based on color segmentation and vertical edge extraction of the original and color segmented images. The network can predict locations of robots, if they are in the foreground of the image and located on green floor. However, two issues have to be discussed for further improvements.

1. The first layer filters have a bias towards red color. This characteristic results from the training set and data base, which mainly contains images of Naos with blue or red jerseys. Since the blue tone of the jerseys is not as distinctive as red, the filters learn a bias towards red jerseys. Additionally, the jersey color is not predefined

anymore in SPL games. Thus, learned filters should not have any bias towards a jerseys color in further developments.

2. The third filter of the first convolutional layer does not produce a distinct output of information but rather a constant output. This leads to co-adaption of filters in the second layer filters. I.e. filters of the subsequent layer rely on the presence of this constant output and therefore learn e.g. more negative filter values. This in turn results in unnecessary computations, less robustness and makes the filters more difficult to analyze. [29] proposes a method called *Dropout* to prevent co-adaption of filters and overfitting which is not further evaluated here but an interesting approach.

4.4 Test Results and Discussion

The original test set of 350 images is extended by horizontal flipping and image section cropping to 21,000 images. The following table depicts the recall and precision for both methods.

Method	**Recall**	**Precision**	**Complete Evaluation**	**Training Time**
BING	0.210	0.004	20 ms	15 min.
DCNN	0.293	0.619	37 ms	160 min.

Tbl. 7: Measured results after evaluation on validation set after training with weight decay ($\lambda_{reg} = 0.01$).

BING has obviously a worse recall and precision ability but needs only half of the complete computational time for an image evaluation. In general both methods suffered from large image inputs. In the case of BING, a large input raises computational time and promotes unnecessary prediction errors which in turn significantly reduces precision and recall (s. section 4.2.2). It is therefore not recommendable as object localization method in the available implementation of [16]. Since the data augmentation techniques were taken from sources related to DCNNs the augmentation was biased towards these. The presented results could thus have been slightly better but the general problem of this approach stays unaffected. The DCNN approach suffered from the input dimensions primarily in terms of computational time. It internally only operates on a $136 \times 136 \times 3$ RGB image but input data is provided as $640 \times 480 \times 3$ RGB image. About $\frac{3}{4}$ of the computational time goes into loading and downsizing the original input. Thus, the implementation cannot be directly transferred to the Nao. Taking into account that image resizing is already performed on the Nao, the proposed network structures in a lower level implementation could be tested on the Nao for further evaluation of computation time. The training duration for the DCNN is more than 10 times longer than for BING but with approximately 165 minutes still maintainable because several training runs can be performed within one day. Furthermore, only CPU computing was used, so that training with a GPU would result in shorter training duration.

In regard to the original project description, the overall results show that the BING

method does not yield sufficiently precise results whereas the DCNN approach needs too much computational time. However, the evaluation of both approaches offered insight into the respective object localization reasoning of each system and also suggestions to amend the stated problems for further implementations and research (s. section 4.2.2 and 4.3.5).

5 Conclusion and Outlook

The goal of this work was to apply a DCNN to the problem of object localization on images in the scenario of robot soccer and analyze the trained system. Another machine learning method, which utilizes two SVMs and normalized gradient images, was applied to the same task for comparison.

Chapter 1 introduced the background and fundamentals of both methods to provide a basic understanding of each. Based on that, chapter 2 described the used approaches with respect to their concept and implementation. Especially the proposed DCNN structure had to be redesigned to meet the limiting hardware constraints. Since training is the essential part of machine learning, chapter 3 points out typical problems, which should be considered in machine learning and suggests solutions to resolve these. Finally, chapter 4 presents both methods in terms of their localization performance and explores the reasoning of each trained system, which lead to the final results.

In the course of this, a suitable network structure was developed and implemented with respect to the hardware constraints. To resolve the general problem of overfitting due to a lack of training data instances, three methods were applied in combination. For this purpose three data augmentation techniques were implemented to extend the number of training samples. The implementation of the image contrast based approach called BING was taken from the respective author and therefore only varied in its hyperparameters. Both methods struggled to meet the predefined limits of the hardware constraints. However, they still offer the potential to be used on the Nao, if re-implemented with respect to the following suggestions. The BING method would benefit from smaller input images because it tries to detect very small objects on the original sized image, which is not necessary in robot soccer. The input dimension reduction would also decrease the processing time and make this approach faster because the input is resized several times during prediction. The DCNN also spend a major part of processing time on resizing the input to the desired dimension. Since the necessary dimension reduction is only performed once before being fed into the DCNN, the structure of the DCNN could be altered to take the already present down scaled images of other modules running on the Nao. The results also showed that the *TensorFlow* library is not suitable for the real application of small scale neural networks, although it offers in combination with *Python* several convenient options to track the learning process. It is therefore recommendable to utilize lower level libraries for a real time application.

A two layer robot classifier was also implemented as a byproduct of this work for the use of weight transfer to the localization network. Unfortunately, it could not be properly evaluated due to limited time constraints. Nevertheless, it showed precision values of up to 70 % and recall of over 50 % on several validation runs. It would therefore be interesting, if the BING approach can be turned into a more selective and precise object localizer for a limited set of objects like it is given in robot soccer. The suggestions stated in the previous chapter could therefore be tested and evaluated in future works. The combination of a fast proposal generator and a small classifier DCNN could yield a very efficient object detection method.

In general the lack of training data for object localization posed a non-negligible problem, which can hardly be performed by computer programs alone. Marking objects on images often requires human expertise and results in very time consuming work. To simplify this, further research could be conducted to implement a multi-class object labeling tool, which could then again be extended to generate object proposals itself and only requires human post-editing. To additionally enhance existing image sets, more data augmentation techniques could be explored in terms of their effect and suitable target application. Different augmentation techniques might be suitable for object classification or localization. DCNNs in general offer multiple parameters to change like the filter size, filter number in each layer, layer number, types of layers, input dimension, activation function etc. just in the case of a DCNN. In this work only the filter size and number were slightly varied and evaluated. Therefore, deeper research could be performed in terms of optimal network layouts for a specific task like classification on the MNIST, CIFAR-10 or CIFAR-100 data set.

References

[1] M. Everingham and J. Winn, "The pascal visual object classes challenge 2012 (voc2012) development kit," 2012.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once - unified real-time object detection," 2015.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," *CVPR*, 2015.

[5] Hubel and Wiesel, Eds., *Receptive Fields and Functional Archetecture of Monkey Striate Cortex*, 1968.

[6] T. C. RoboCup, "Robocup standard platform league (nao) rule book," 2016.

[7] Aldebaran Robotics, "Nao software 1.14.5 documentation: Nao technical overview," 2016. [Online]. Available: http://doc.aldebaran.com/1-14/family/robots/index_robots.html

[8] E. Alpaydın, *Introduction to machine learning*, third edition (online-ausg.) ed., ser. Adaptive computation and machine learning. Cambridge, Massachusetts and London, England: The MIT Press, 2014. [Online]. Available: http://site.ebrary.com/lib/alltitles/Doc?id=10919034

[9] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, C. Suen, A. Coates, A. Maas, A. Hannun, B. Huval, T. Wang, and S. Tandon, "Unsupervised feature learning and deep learning: Deep learning tutorial," 2016. [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/

[10] F.-F. Li, A. Karpathy, and J. Johnson, "Cs231n: Convolutional neural networks for visual recognition," 2016. [Online]. Available: http://cs231n.github.io/neural-networks-1/

[11] H. Werner, "Neural and genetic computing for control engineering."

[12] A. Krizhevsky, I. Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012.

[13] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaption of feature detectors," 2012.

[14] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks: Microsoft word - ijae_26 3," *International Journal of Artificial Intelligence and Expert Systems*, no. 4, 2010.

[15] M. Everingham, L. van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[16] Ming-Ming Cheng, Niloy J. Mitra, Xiaolei Huang, Philip H. S. Torr, and Shi-Min Hu, "Global contrast based salient region detection," 2014.

[17] R.-R. Grigat, "Pattern recognition."

[18] Primate Labs Inc., "Linux pc (intel atom z530)," 2016. [Online]. Available: http://browser.primatelabs.com/geekbench2/306646

[19] P. F. Felzenszwalb, R. B. Girshick, McAllester David, and D. Ramanan, "Object detection with discriminatively trained part based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[20] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "liblinear a library for large linear classification," *Journal of Machine Learning Research*, vol. 2008, no. 9, 2015.

[21] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CVPR*, 2014.

[22] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov, "Scalable object detection using deep neural networks," 2014.

[23] M. Everingham and J. Winn, "The pascal visual object classes challenge 2007 (voc2007) development kit," 2007.

[24] G. Dreyfus, *Neural Networks: Methodology and Applications*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2005. [Online]. Available: http://dx.doi.org/10.1007/3-540-28847-3

[25] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Baiduvision: Scaling up end-to-end image recognition: arxiv:1501.02876," *arXiv preprint*, 2015.

[26] Aldebaran Robotics, "Nao datasheet," 2016. [Online]. Available: https://www.ald.softbankrobotics.com/sites/aldebaran/files/nao_datasheet.pdf

[27] Maklaan, "Rgb cube: Maklaan [cc by-sa 3.0 (http://creativecommons.org/licenses/by-sa/3.0)]," 2016. [Online]. Available: https://commons.wikimedia.org/wiki/File%3ARGB_color_cube.svg

[28] D. P. Kingma and J. L. Ba, "Adam - a method for stochastic optimization," *International Conference for Learning Representations*, 2015.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout - a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

A Appendix

A.1 Evaluation

A.1.1 Weight Decay

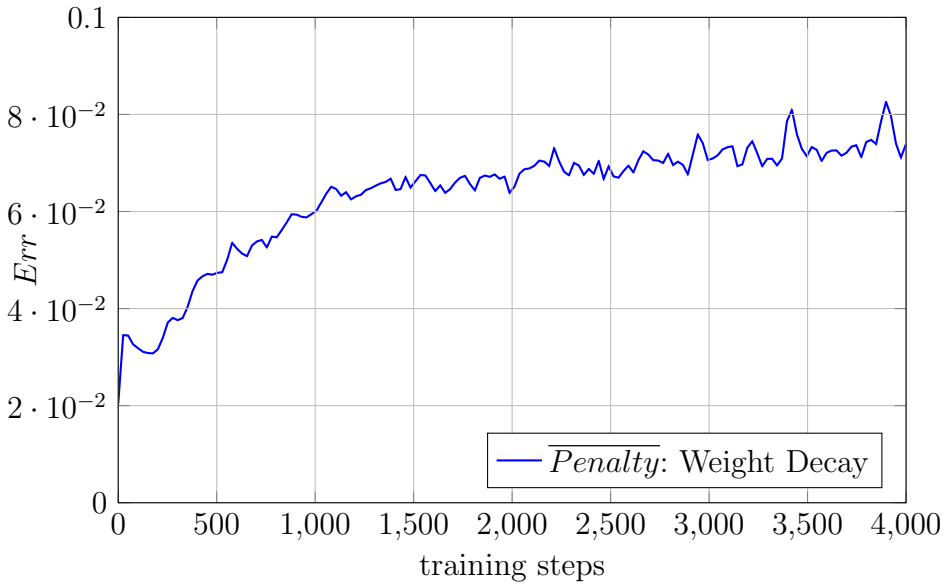


Fig. A.1: Regularization loss during regularization test in chapter 4.

A.1.2 Data Augmentation

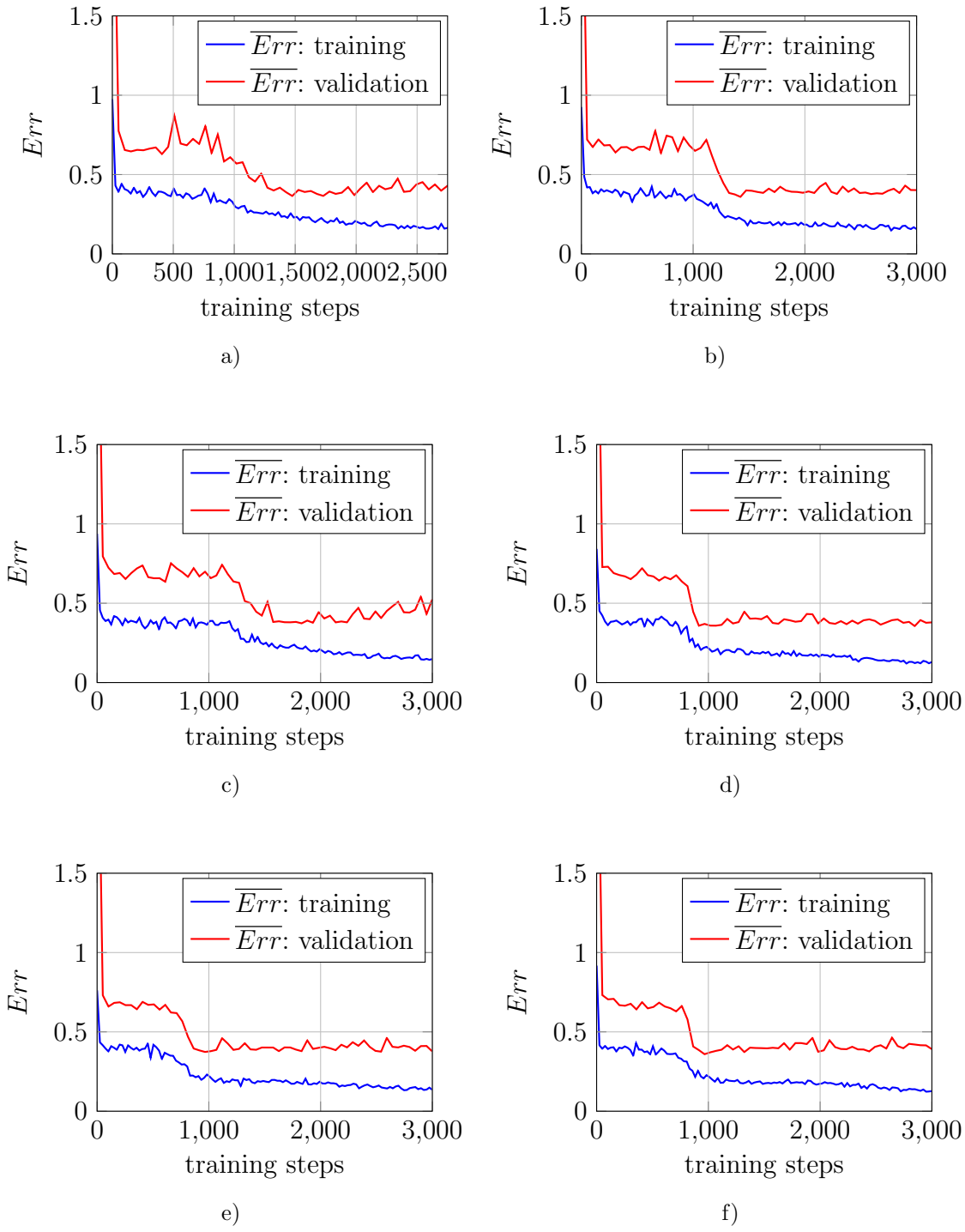
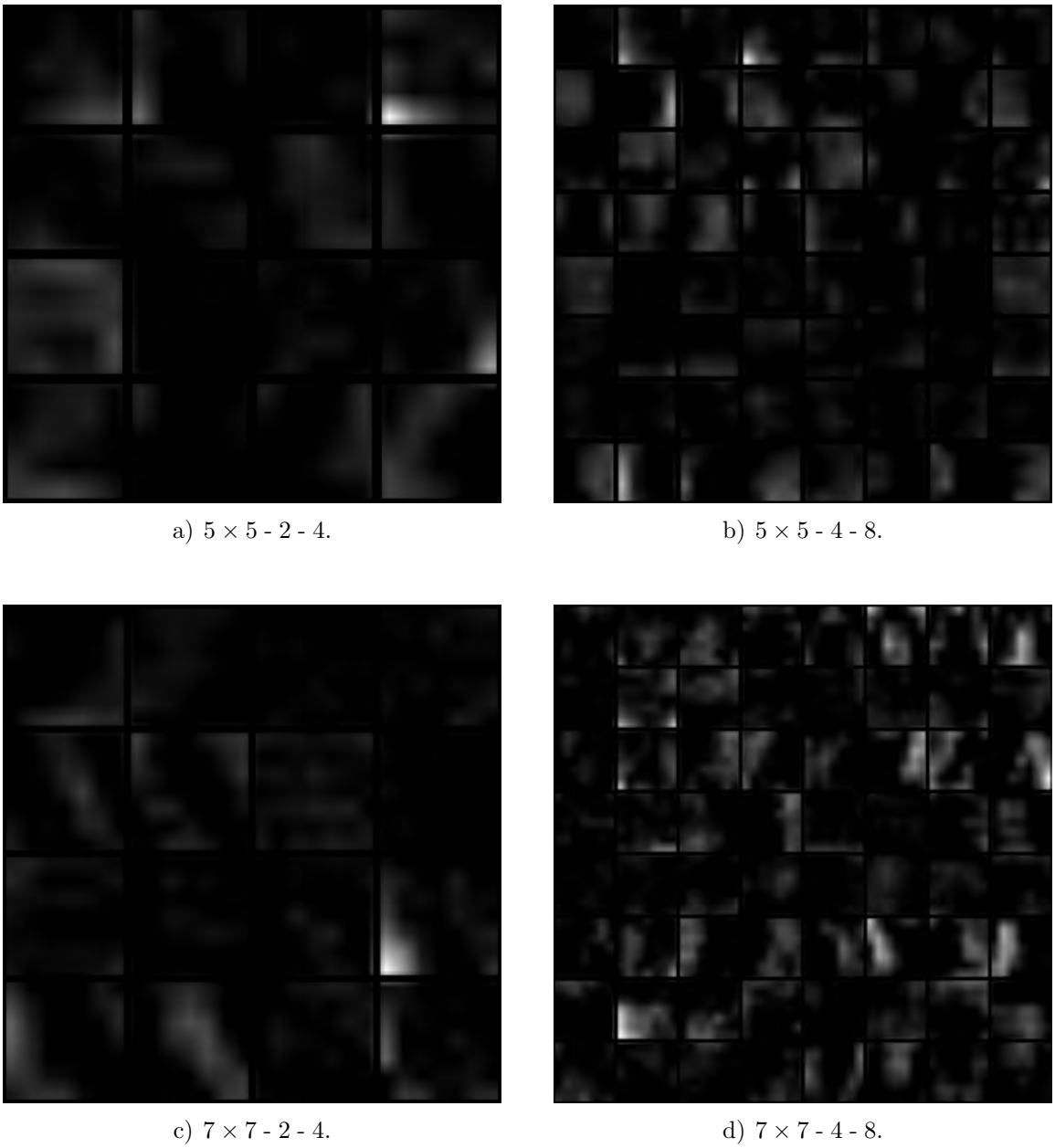


Fig. A.2: Training and validation error progression during training of HSV color model augmentation test. a) - c) without targeted augmentation, d) - f) with targeted augmentation.

A.1.3 Model Selection (DCNN)



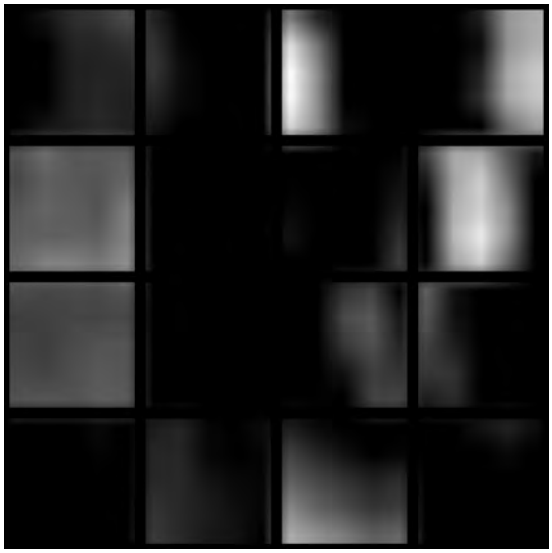
a) 5×5 - 2 - 4.

b) 5×5 - 4 - 8.

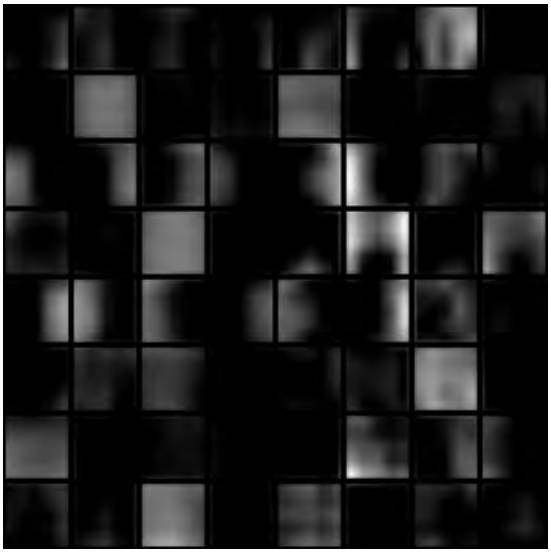
c) 7×7 - 2 - 4.

d) 7×7 - 4 - 8.

Fig. A.3: Plot of second layer filters for each network structure. Learned without weight decay.

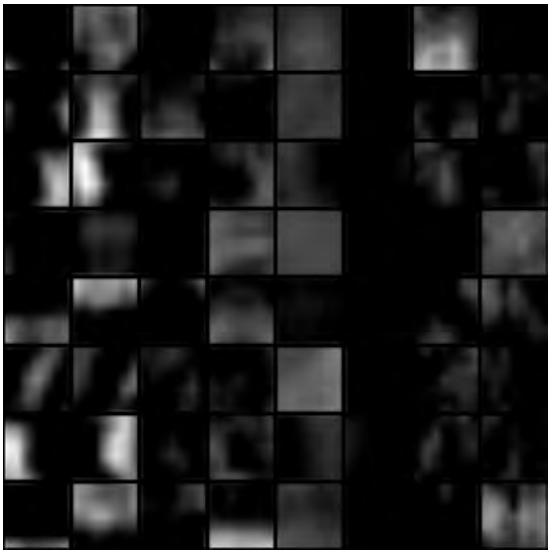


a) 5×5 - 2 - 4.



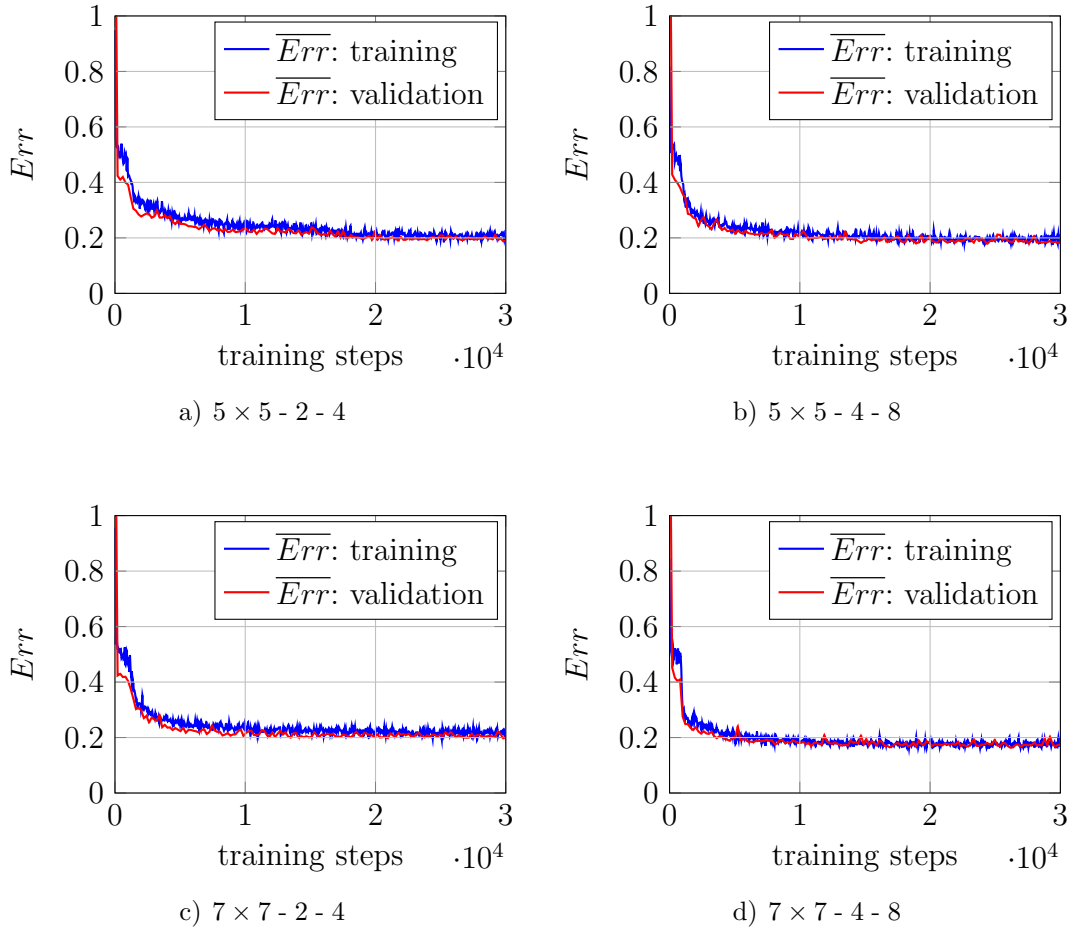
b) 5×5 - 4 - 8.

c) 7×7 - 2 - 4.



d) 7×7 - 4 - 8.

Fig. A.4: Plot of second layer filters for each network structure. Learned with weight decay.



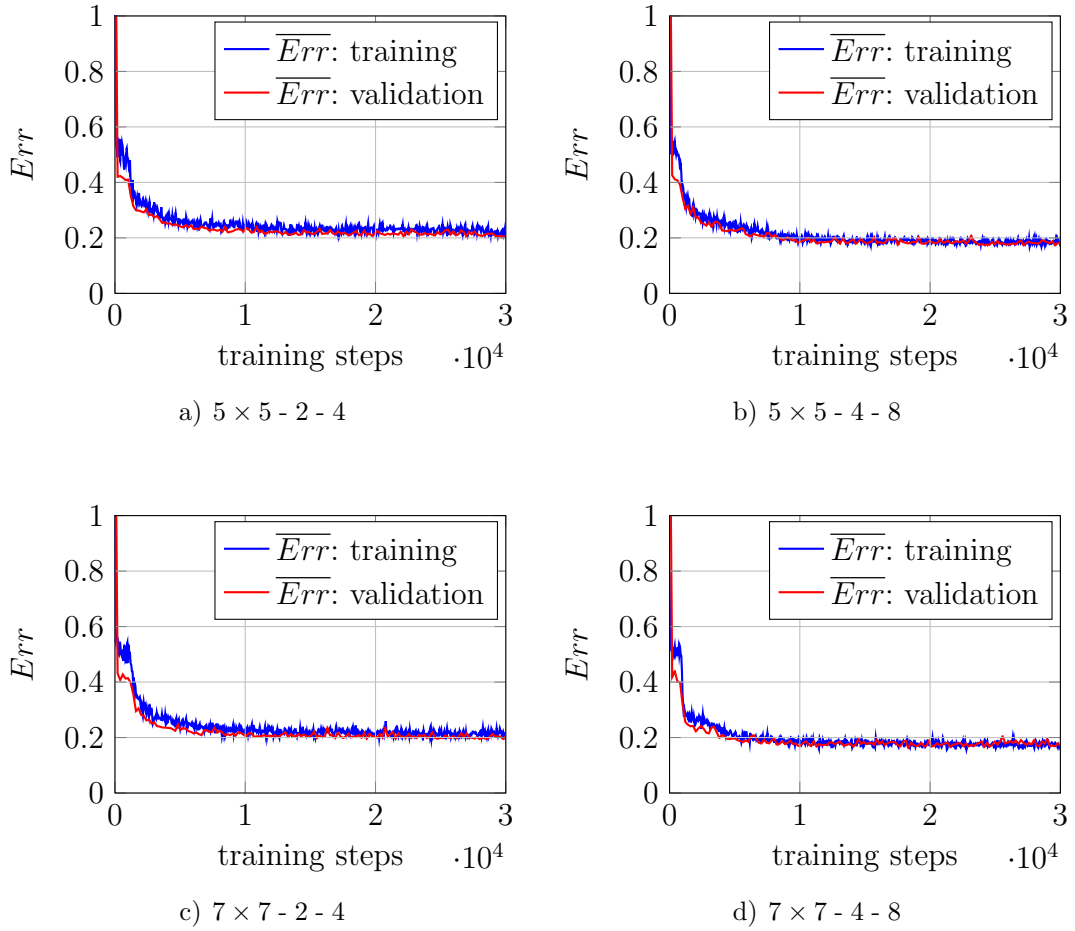
a) 5×5 - 2 - 4

b) 5×5 - 4 - 8

c) 7×7 - 2 - 4

d) 7×7 - 4 - 8

Fig. A.5: Error progression of DCNN model selection during first training run with weight decay.



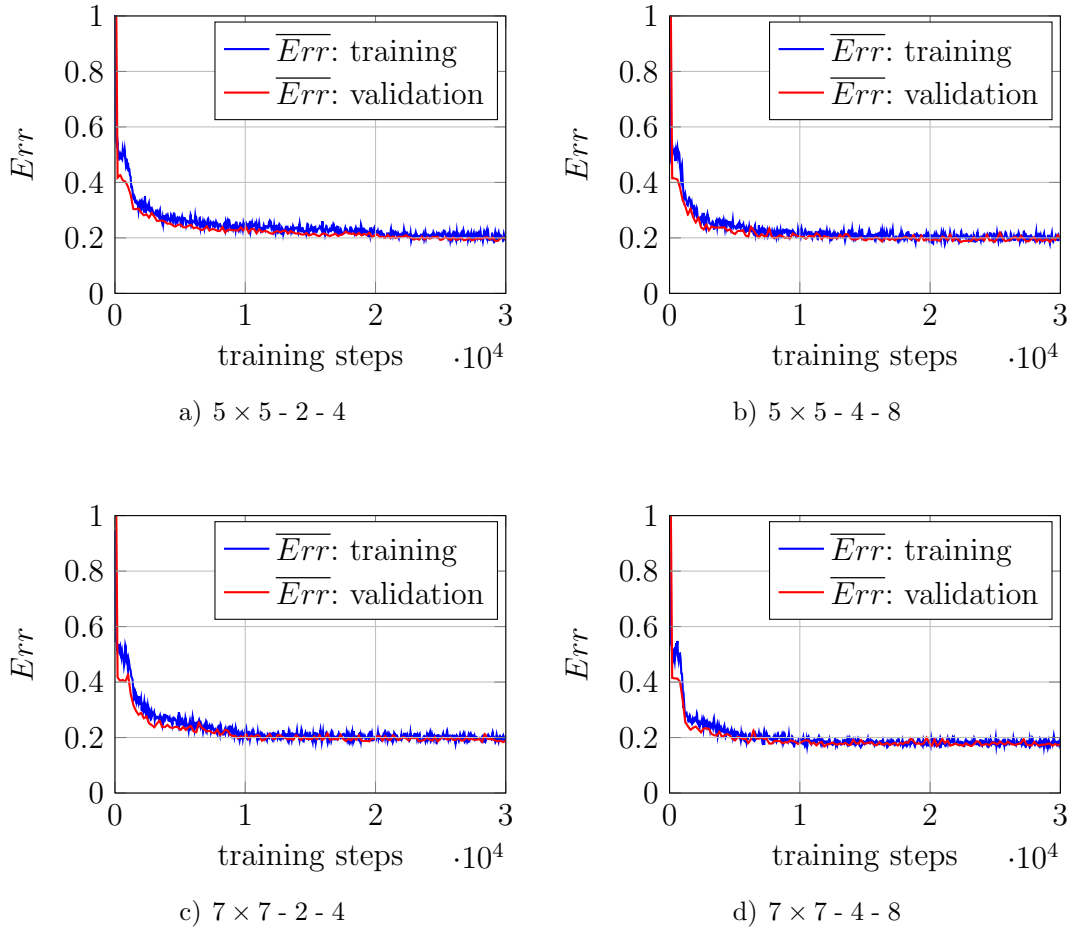
a) 5×5 - 2 - 4

b) 5×5 - 4 - 8

c) 7×7 - 2 - 4

d) 7×7 - 4 - 8

Fig. A.6: Error progression of DCNN model selection during second training run with weight decay.



a) 5×5 - 2 - 4

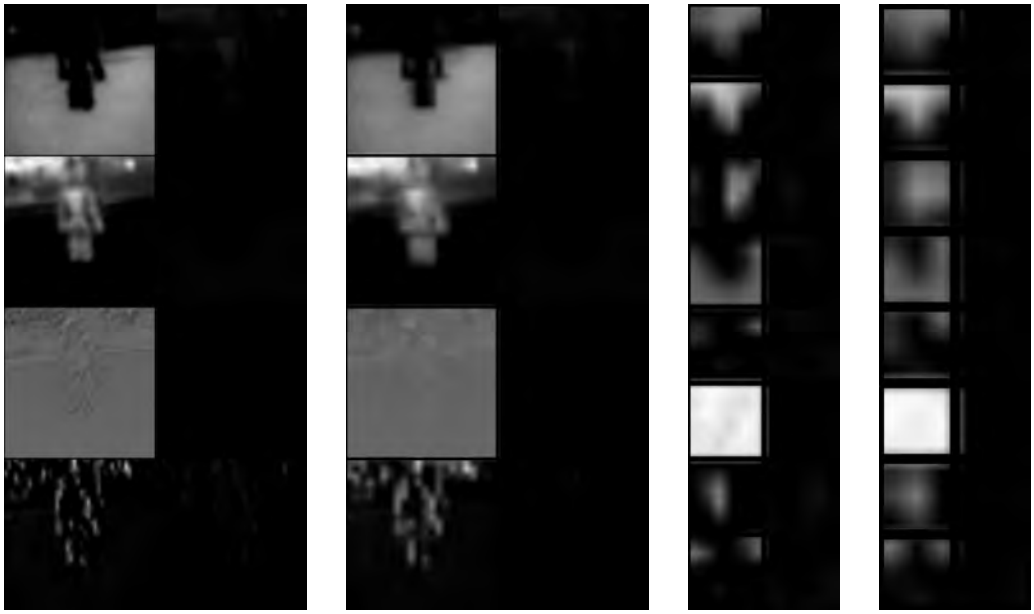
b) 5×5 - 4 - 8

c) 7×7 - 2 - 4

d) 7×7 - 4 - 8

Fig. A.7: Error progression of DCNN model selection during third training run with weight decay.

A.1.4 Model and Filter Evaluation



a) Output of first convolutional layer.

b) Output of first *max-pooling* layer.

c) Output of second convolutional layer.

d) Output of second *max-pooling* layer.

Fig. A.8: Plot of inter layer outputs for example image in filter analysis section.