Technische Universität Hamburg-Harburg

Vision Systems

Prof. Dr.-Ing. R.-R. Grigat

Optimal CNN Hyperparameters for Object Detection on NAO Robots

Master Thesis

Georg Christian Felbinger

04.06.2018

First Examiner: Prof. Dr.-Ing. R.-R. Grigat

Second Examiner: Prof. Dr.-Ing. T. Knopp

Author's Declaration

I, Georg Christian Felbinger, born on 17.11.1989 in Gunzenhausen, hereby declare on oath that I compiled this master thesis, submitted to Technische Universität Hamburg-Harburg (TUHH), on my own. I only used the declared sources and auxiliaries.

Hamburg, 04.06.2018 Georg Christian Felbinger

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic systems are increasingly becoming an integral part of everyday life and are already indispensable in many areas today. While the first industrial robots are already in use for almost a decade, autonomously operating robotic systems are only used in the recent research and industry. Since 1997, the RoboCup has been organizing an annual competitions of autonomous systems in different disciplines.

This thesis is written in the context of the RoboCup Team HULKs of the Hamburg University of Technology (TUHH), which is participating in the Standard Platform League (SPL). This chapter provides a brief overview about the RoboCup and the SPL, as well as the used robotic system NAO and explains the motivation and goals of this thesis.

1.1 The RoboCup

In May 1997, when IBM Deep Blue defeated the human world champion Garry Kasparow in chess, forty years of challenge in the AI community came to a successful conclusion [1]. On July 4, 1997, NASA's MARS Pathfinder mission made a successful landing and the first autonomous robotics system, Sojourner, was deployed on the surface of Mars [2]. Together with these accomplishments, RoboCup was founded in 1997. The idea was to build robots who are able to play soccer and beat the human World Cup champion team by 2050 [3]. Today, the RoboCup consists of various soccer leagues, as well as competitions for housekeeping, rescue or industrial robots.

1.2 The NAO Robotic System

The NAO Robotic System, hereafter referred as NAO, is a 58cm high humanoid robot from SoftBank Robotics. It is continually developed since 2006 and currently available in the fifth version [4], an example can be seen in 1.1. The NAO is used by the RoboCup SPL Teams as well as in this thesis.

Figure 1.1: *Two HULKs NAO robots defending against BHuman at the RoboCup German Open 2018.*

The computer in the NAO is based on a 1.6 GHz Intel Atom Z530 and 1 GB RAM. It has two cameras, each with a maximum resolution of 1288x968 pixels and a field of view of 72.6° [5]. The perceptional fields of the cameras overlap only partially, hence no stereoscopic vision is possible.

1.3 The Standard Platform League (SPL)

As mentioned, there are several soccer leagues in the RoboCup, each working on a different goal of research. One of those is the Standard Platform League (SPL) where all participating teams use the same robot, namely the NAO robot. As every team uses this platform, this league is focused on implementing fast and robust software for real-time soccer purposes.

The NAO robots play fully autonomously and each one takes decisions separately from the others. Each team consists of five robots, playing together using communications. According to the rules, the games are played on a green field of artificial turf of size 9×6m with white lines and goal posts, with no other landmarks [6]. To improve the league every year, the rules of the SPL games get continuously adapted towards those of a real soccer game. Since 2016, the ball is a realistic white and black soccer one [7].

1.4 Motivation

With the success of the HULKs in recent years demands on the world model increased alongside. Until now, the robots perception of the outside world consists of a line and ball detection from the camera image as well as near field obstacles using sonar. A robot detection is necessary to be able to react to opponent robots for strategic decisions.

Approaches based on convolutional neural networks for object detection lead to promising results in previous work [8], [9], [10]. These methods save a lot of work, as no manual feature extraction is necessary.

One drawback of neural network based methods is the large amount of computation time during the inference. To reduce the overall expense of the object detection, a multi-class detector which is able to perceive balls and robots at the same time is desirable.

Another problem is the large amount of hyperparameters for such classifiers. Our recent work already produced useful results using genetic algorithms for optimization [8]. With increasing amount of hyperparameters the effort of executing a genetic algorithm on such a search space highly increases. In the area of game theory, there are many algorithms designed to handle such large search spaces. To make use of game tree search algorithms, the problem of designing a multi-class classification network has to be formulated as a board filling game. The central idea of this thesis is to develop a multi-class object detection framework for the NAO robot using optimized convolution neural networks. For the optimization step an Monte Carlo tree search based method is developed and compared against genetic approaches.

1.5 Goals

The aim of this thesis is to develop a multi-class object detector to perceive balls and robots at the same time on the NAO robotic system. A framework for collecting and labeling data is to be implemented. Based on this data a region of interest search is needed to find areas on the image that may contain the objects. Those candidates alongside with the labeled ground truth can then be used to train a CNN classifier. By formulating the design of this classifier as a board filling game, it can be optimized using a game tree search algorithm. Therefore, a Monte Carlo tree search based framework will be implemented and compared against existing genetic approaches.

The resulting classifier should be able to achieve a similar classification performance on the ball class as the current one [8, p. 35]. As there is no robot detection in the HULKs software framework yet, the classification performance should build a descent baseline.

1.6 Related Work

In the field of neural network research, there has been a lot of interest in automatically learning network architectures recently. [11] proposes a method using genetic algorithms for evolving the architectures and connection weight initialization values of a deep convolutional neural network to address image classification problems. Recent research within the HULKs team demonstrate application for optimizing hyperparameters for convolutional neural networks in real-time applications for single class problems [8], [12].

Beside genetic or reinforcement learning approaches, game tree search based methods became popular recently. [13] proposes a method for learning the structure of CNNs using a sequential model-based optimization strategy, searching for structures in order of increasing complexity. [14] describes a framework for automatically designing and training deep models using Monte Carlo tree search and sequential model-based optimization, outperforming random search. An extension for solving algorithm configuration is shown by [15] for general algorithm configuration problems, allowing many categorical parameters and optimization for sets of instances.

Within the field of RoboCup there is currently only one team known to use a multi-class object detector on the NAO. The approach of NAO Team HTWK achieves a recall of 50% of all robots, 93% of balls and 78% of penalty spots - with an overall precision higher than 99%. [16]

1.7 Problem Overview and Thesis Structure

The prerequisites chapter explains the basic terms, used software and the algorithm which was developed as a part of this thesis. The data acquisition chapter shows, how the data used for learning the models was generated, labeled and set up.

After the data is set up properly, the network architecture can be optimized for the problem using an optimization algorithm, as described in chapter model training and optimization. The evaluation of the hyperparameter optimization can be found in chapter evaluation. This also covers the comparison to previous work on hyperparameter optimization as well as the inference of the resulting network on the NAO.

Finally, chapter conclusion and outlook summarizes the thesis and gives an outlook to possible future work.

Chapter 2

Prerequisites

In this chapter, the basic terms for this thesis are explained. It also includes the used software and libraries as well as a brief description of the Monte Carlo tree search, artificial neural networks and convolutional neural networks.

2.1 Basic Terms

There are many common terms in the field of machine learning and artificial intelligence. As this thesis focuses on MCTS optimization and convolutional neural networks, these terms itself and related ones are explained.

2.1.1 Metrics

The classifier, which maps an example image to a class, is evaluated using the following metrics. For each class c, the set of positives p_c is the set of example candidate images containing an object of this class. The other candidate images are assigned to the set of negatives n_c.

True positives (tp_c) are those examples from the setup p_c, which were assigned to class c by the classifier. The true negatives (tn_c) are those examples from the sets n_c, which were not assigned to an object of class c, respectively. False positives / false negatives (fp_c / fn_c) are the images from p_c / n_c, which were assigned to the wrong class.

The true positive rate of a class c is the relative amount of correctly assigned examples within the set p_c: $TPR_c = \frac{|tp_c|}{|p_c|}$. The true negative rate is analog: $TNR_c = \frac{|tn_c|}{|n_c|}$.

2.1.2 *k*-Fold Cross-Validation

In k-fold cross-validation, sometimes called rotation estimation, the dataset D is randomly split into k mutually disjoint subsets (the folds) $D_1, D_2, ..., D_k$ of approximately equal size. The classifier is trained and tested k times; each time $t \in \{1, 2, ..., k\}$ it is trained on $D \setminus D_t$ and tested on D_t [17, pp. 2–3]. Figure 2.1 illustrates a 4-fold cross validation.

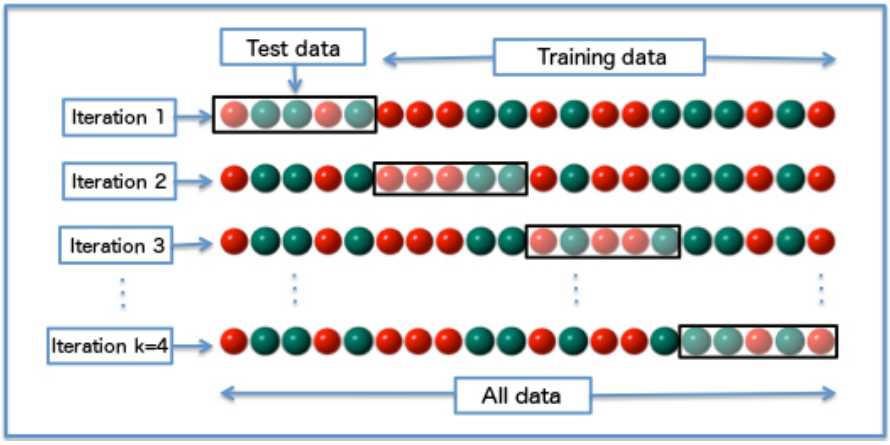


Figure 2.1: *Diagram of k-fold cross-validation with $k = 4$. [18]*

2.2 Used Software

2.2.1 HULKs NAO Framework

The HULKs NAO Framework is the software which runs on the NAO during SPL games, written in C++. It can also be used for testing and debugging purposes, together with the MATE tool. Beside the NAO, it has multiple more compile targets, i.e. running the software with SimRobot.

2.2.2 HULKs MATE

The HULKs framework features a debugging and configuration interface called MATE. It is possible to export variables (including images) with an associated name so that they can be sent to a PC or written to a log file. It is able to connect to a running instance of the HULKs framework via a TCP or Unix socket and provides a user interface. Data can be displayed in several types of views listed in table 2.1.

Table 2.1: *Debugging Views of MATE.*

View	Description
Text	Prints arbitrary data structures in JSON format.
Image	Displays JPEG compressed images that are sent by the NAO.
Config	Interface to query and edit configuration values at runtime on the connected target
Plot	Views plots of a numeric value over time. Multiple variables can be plotted color-coded in the same panel. The value range can be determined automatically and the number of samples can be adjusted.
Map	View to display various position related information such as robot poses or ball positions on a two dimensional area of arbitrary size, e.g. a SPL field.

A specific setup of views can be stored and loaded as file to allow for fast access to often used view combinations.

2.2.3 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows to deploy computation to one or more CPUs or GPUs on a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. [19]

2.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are computing systems inspired by the neurons of a natural brain. Inputs are fed into a network of neurons which processes the information. Figure 2.2 shows the structure of an ANN.

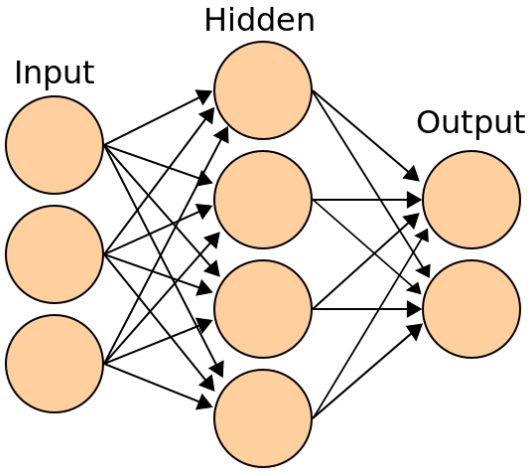


Figure 2.2: *An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another. [20]*

2.3.1 Neurons

As in a natural brain, the computation is done by a composition of single neurons. Figure 2.3 shows the functionality of those.

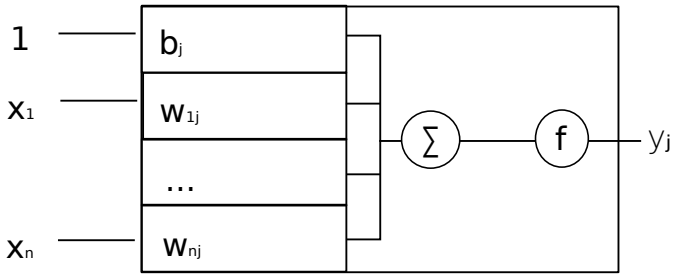


Figure 2.3: *The functioning of a single neuron j. First, the input vector gets element-wise multiplied by the weight vector. Afterwards the bias is added to the sum of the resulting vector. Finally, the value is applied to the activation function, yielding the output of the neuron.*

Given a layer of k neurons with input vector $x \in \mathbb{R}^n$, weight matrix $W \in \mathbb{R}^{k \times n}$ and bias vector $b \in \mathbb{R}^k$. The result of neuron j can be computed as $y_j = f(W_{.j}x^T + b_j)$, where

$W_{.j}$ is the j-th column of W. Hence a whole layer of neurons can be computed as $y = f(Wx^T + b)$.

2.3.2 Activation Functions

In biology, a neuron "fires" in a non-linear way depending on the input [21, p. 1065]. In ANNs, this is achieved by applying an activation function to every value of a layers output. In this thesis, the following activation functions are investigated.

Hyperbolic Tangent (TANH)

The hyperbolic tangent is a S shaped function which maps the input to the range $[-1,1]$. It can be used as a differentiable model for binary output.

$$TANH(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.1}$$

Rectified Linear Unit (ReLU)

The ReLU function has become one of the most popular activation functions for neural networks, due to the computation simplicity. It maps every positive input to itself, while dropping all negative inputs to zero.

$$ReLU(x) = \begin{cases} x & ,x > 0 \\ 0 & ,otherwise \end{cases} \tag{2.2}$$

2.4 Convolutional Neural Networks

CNNs have become very popular in recent years for object detection in images. It is the leading approach in various benchmark datasets. For example, in the MNIST dataset, CNN based approaches achieve the best results with a test error rate of down to 0.23 [22]. In the CIFAR-10 dataset, CNNs achieved a accuracy of up to 96.53% [23].

They extend ANNs by adding convolutional, pooling and normalization layers before fully connected layers which then calculate the final output.

2.4.1 Convolutional Layer

Convolutional layers apply a trainable convolution mask on the input. In this thesis, only 2-dimensional convolutions are used, meaning a input image with q channels is mapped to an output image with k channels. Equation eq. 2.3 shows the computation of a convolutional layer.

$$y_{i,j,k} = \sum_{di,dj,q} x_{i+di,j+dj,q} \cdot m_{di,dj,q,k}$$
$$x \in \mathbb{R}^{i \times j \times q}, m \in \mathbb{R}^{di \times dj \times q \times k} \tag{2.3}$$

2.4.2 Pooling Layer

Pooling layers are used for downsampling images between convolutional layers. They reduce every dimension of every image channel by applying a reduce function to neighbouring pixels. This work focus on $\max(a,b,c,d)$ (maximum value of arguments) and $\text{avg}(a,b,c,d)$ (arithmetric mean of arguments) as pooling functions. Figure 2.4 demonstrates a 2x2 max-pooling layer for a single channel image.

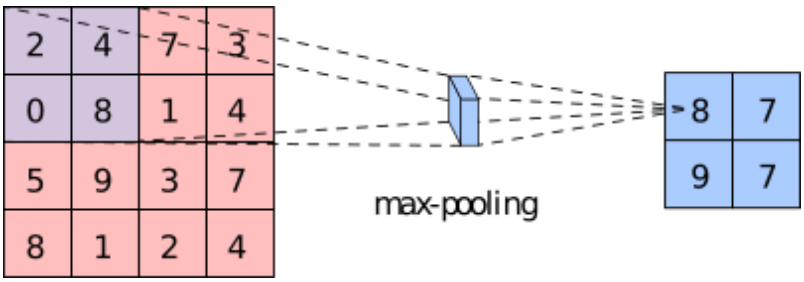


Figure 2.4: *Schematic drawing of a 2x2 sized max-pooling layer downsampling an 4x4 input to 2x2 with a stride of 2 so that no overlapping occurs. [24]*

2.4.3 Local Response Normalization

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, it can still be found that the following local normalization scheme aids generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x,y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression [25, p. 4]:

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=\max(0,\frac{i-n}{2})}^{\min(N-1,\frac{i+n}{2})} (a^j_{x,y})^2\right)^\beta} \tag{2.4}$$

where the sum runs over n adjacent kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, β are parameters whose values are chosen by the recommondations of [26].

2.5 Monte Carlo Tree Search

Monte Carle Tree Search (MCTS) is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results [27]. It has already had a profound impact on artificial intelligence approaches for domains that can be represented as trees of sequential decisions such as games.

MCTS uses two policies to build the search tree, namely the tree and the rollout or default policy [14]. The tree policy determines the path to be traversed from the root to a bottom node of the already expanded tree. The path from the leaf of this bottom node to a leaf is then determined by the rollout policy. A leaf node can be evaluated to the true result, e.g. by training and evaluation of a resulting network. The score is used to update the statistics of the nodes in the current path, starting from the root. Each node in the expanded tree keeps statistics about the number of times it was visited and the average score of the models that were evaluated in the subtree at that node.

2.5.1 Algorithm

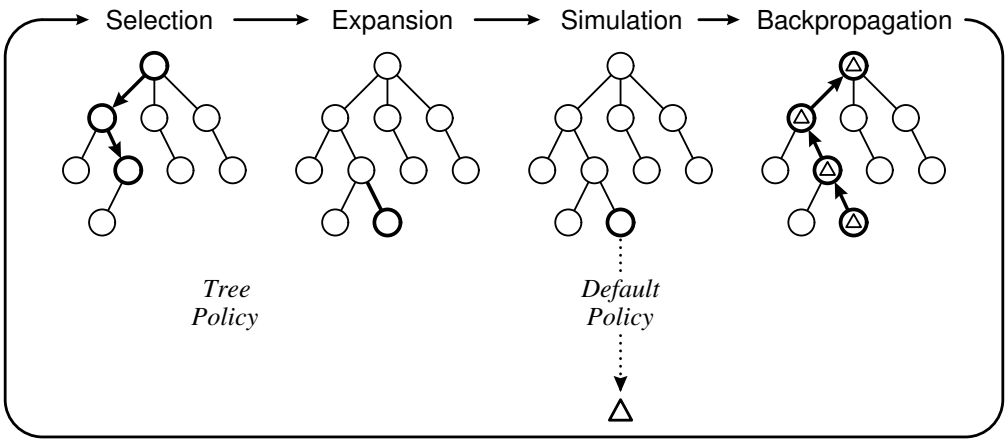


Figure 2.5: *One iteration of the MCTS algorithm. [27, p. 6]*

The MCTS algorithm works in an iterative way and yields the current best play estimate after each iteration. figure 2.5 shows one iteration of the basic MCTS algorithm. Each node in the search tree represents a state of the domain, and directed links to child nodes represent actions leading to subsequent states. Every iteration consists of the following steps: Selection, Expansion, Simulation and Backpropagation. Algorithm 1 summarizes those steps.

Algorithm 1 General MCTS approach. v_0 is the root node corresponding to state s_0. v_l is the last node reached during the tree policy stage and corresponds to state s_l. The reward from the terminal state reached by the rollout policy from state s_l is assigned to Δ. $a(bestChild(v_0))$ is the action a that leads to the best child of the root node v_0.

```
1: function MCTS(s_0)
2:     v_0 ← createRootNode(s_0)
3:     while keepRunning() do
4:         v_l ← treePolicy(v_0)
5:         Δ ← rolloutPolicy(s(v_l))
6:         propagate(v_l, Δ)
7:     end while
8:     return a(bestChild(v_0))
9: end function
```

Selection

Starting at the root node, the tree policy is recursively applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a non terminal state and has unvisited (i.e., unexpanded) children.

Upper Confidence Bound The most popular algorithm in the MCTS family is the upper confidence bound for trees (UCT). It is very simple and efficient and guaranteed to be within a constant factor of the best possible bound on the growth of regret. It is thus a promising candidate to address the exploration–exploitation dilemma in MCTS: every time a node (action) is to be selected within the existing tree, the choice may be modeled as an independent multi-armed bandit problem. A child node j is selected to maximize

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{2.5}$$

where n is the number of times the current (parent) node has been visited [27]. n_j is the number of times child j has been visited and $C_p > 0$ is a constant. Ties are decided randomly.

Sequential Model Based Optimization Model-based optimization methods construct a regression model that predicts performance and then use this model for optimization. Sequential model-based optimization (SMBO) iterates between fitting a model and gathering additional data based on this model. In the context of parameter optimization, the model is fitted to a training set

$$S = \{(\Theta_1, o_1), (\Theta_n, o_n)\}$$

where Θ_i is a complete instantiation of the target algorithm parameters and o_i is the performance of the algorithm using the parameters Θ_i. [15, p. 3] Using such a model as tree policy enables the possibility for the MCTS to share information between nodes other than through common ancestors. This can be addressed by introducing a surrogate function [14] which can be used to capture relationships between models and how promising it is to evaluate any specific model. As the optimization of the tree policy exceeds the scope of this thesis, this is left for future work.

Expansion

One (or more) child nodes are added to expand the tree, according to the available actions. The available actions are determined by the game rules.

Simulation

A simulation is run from the new node(s) according to the default policy to produce an outcome.

Backpropagation

A score is determined based on the result of the simulation and is propagated back to the root node by updating the statistics of the nodes on the path.

2.6 Genetic Algorithm

Genetic algorithms (GAs) were already evaluated in previous work [8]. They led to promising results for optimizing CNNs hyperparameters and will be used as benchmark for the MCTS optimization.

The method used in this work follows a genetic algorithm pattern [28]. The basic elements are chromosomes or individuals $c \in S$, a possible solution in given k-dimensional search space S. The algorithm works in an iterative manner with a fixed number of generations N. A set of n chromosomes used during iteration j is called population $P_j = \{c_1, ..., c_n\} \subset S$. The initial population P_0 is generated randomly. In each generation $j \in [1, N]$ the population of the previous iteration is evaluated using a fitness function $f(c) : \mathbb{C}^k \mapsto \mathbb{R}$. Given the previous population and its individuals' fitnesses a set S_j is selected from P_{j-1} as parents. In the next step a mutation function will be applied to every element in S_j. Finally, mutated parent elements are recombined yielding next generation P_j.

2.6.1 Selection

The selection is done using the following steps. According to a given clipping parameter $c \in [0, 1]$ individuals in the lower c percentile are dropped. The minimal fitness within the population is given by $\min_{k \in [1,n]}(f(c_k))$. Given the other m individuals the probability of surviving is calculated by equation (2.6).

$$p(c_i) = \frac{f(c_i) - \text{minscore}}{\sum_{j=1}^{m} \left(f(c_j) - \text{minscore}\right)} \tag{2.6}$$

Hence, the individual with the lowest fitness value is assigned to the survival probability zero. According to this distribution, n elements are sampled for mutation and reproduction.

Elitism

A common variant of GAs is to allow one or more of the best organisms from the current generation to carry over to the next, unaltered. This strategy is known as elitist selection and guarantees that the quality of the best solution in each generation monotonically increases over time. Without elitist selection, it is possible to lose the best chromosome due to stochastic errors [29, p. 2].

2.6.2 Mutation

For every value within chromosome c a new value will be sampled based on a given mutation probability p_m. If a gene is to be replaced a new random value is chosen.

2.6.3 Reproduction

In the reproduction phase the selected and mutated chromosomes are pairwise randomly sampled. Each pair yields two new children. For every value within the chromosome of a child, the corresponding parent value is chosen randomly.

Chapter 3

Data Acquisition

This Chapter describes the collection and setup of the data. Section 3.1 describes how the raw data was collected from the robot. The raw data was then labeled with ground truth and integrated into the framework as described in section 3.2. Section 3.3 shows how the region of interest search was developed and logged together with the ground truth. Given those results, datasets for training and testing are generated and set up as shown in section 3.4.

3.1 Collection of Images and Metadata

In our previous work the candidate generation was implemented directly within the framework. The results were logged to disk directly on the robot. There are multiple drawbacks of this method. This way, all data which is already collected becomes useless as soon as the candidate generation changes. Also, the real detection rate cannot be determined as there is information about how many objects were missed.

Therefore a framework for collecting data has to be reworked. A module was developed which receives the image and sensor data as well as metadata like which buttons were pressed. Within a fixed rate, the module checks whether a frame can be recorded. Algorithm 2 describes a single cycle of this module.

The collected frames can be used by the HULKs framework. This is done by a virtual robot interface, which reads the recorded frames as sensor data.

Algorithm 2 ReplayRecorder::cycle

```
1: if !allDepenciesValid() then
2:     return                                   ▷ Only record if the data is available
3: end if
4: if bodyPose->fallen then
5:     return                                   ▷ Do not record in fallen state
6: end if
7: if onlyRecordWhilePlaying_() && gameControllerState_->state != GameS-
   tate::PLAYING) then
8:     return                                   ▷ Only record while playing if configured
9: end if
10: if gameControllerState_->penalty != Penalty::NONE then
11:     return                                  ▷ Only record when unpenalized
12: end if
13: if writeThreadBusy_ then
14:     return                                  ▷ Only record when currently not busy
15: end if
16: if currentFrame.camera == imageData_->camera then
17:     return                                  ▷ Only record when camera is different to last one
18: end if
19: if secsSinceLastFrame() < minSecBetweenFrames_() then
20:     return                                  ▷ Only record when timeDiff to last log is high enough
21: end if
22: startRecordFrame();
```

3.2 Labeling of Raw Data

In order to set up a proper train and test data set, the ground truth for the given data is needed. Therefore every object on every collected image which is to be trained needs to be annotated. To do this in a collaborating manner, a web based tool called ‘annotate’ [30] is developed. It is divided into a server and a client part. The labelled data can be integrated into the replay framework mentioned in the last section.

3.2.1 Server

The annotate server REST API ([31], [32]) is written in TypeScript [33]. It is used to handle a database of images together with their annotations. It serves the following interfaces:

Table 3.1: *Overview of the annotate server interface.*

Interface	Method	Description
/v1/auth	POST	Request an OAUTH [34] authentication token for highscores
/v1/auth/clientId	GET	Request the client id given a token
/v1/image	GET	Request a random image from the database
/v2/image	GET	Request a random or specific image from the database
/v1/image	POST	Send the label data for an image to the server using absolute image coordinates
/v2/image	POST	Send the label data for an image to the server using relative image coordinates
/v1/stats	GET	Get the current amount of labeled / remaining images in the database
/v1/labels	GET	Get the label types which need to be collected
/v2/labels	GET	Get the label types for whole image and bounding boxes
/v3/labels	GET	Get the label types for whole image, bounding boxes and lines
/v1/scores	GET	Get the current high score list

3.2.2 Client

The client side of annotate is written in TypeScript [33] using the following libraries:

Table 3.2: *Libraries used by the annotate client.*

Library	Description
axios [35]	REST [31] client for querying the server
react [36]	Framework for User Interfaces
bootstrap [37]	Framework for User Interface Design
redux [38]	State container framework for JavaScript Applications

The client provides an easy to use and very fast way to mark objects on the images such as lines and boxes. Beside the objects, a label for the whole scene can be applied in order to sort out images which are not useful for the training set. Figure 3.1 illustrates the user interface of the annotate client. For this thesis, over 20000 images where annotated

with robots and balls.[1]

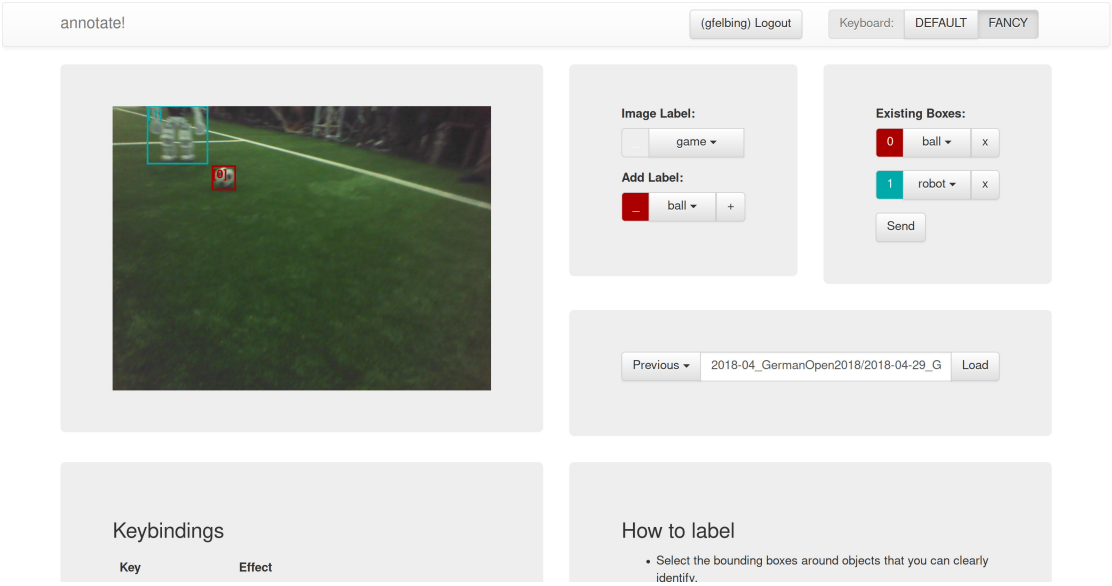


Figure 3.1: *User Interface of the annotate[30] client. Header: User Login for High-scores, Selection of the keyboard layout; Top Left: An annotated image containing a robot and a ball. Top Center: The selector for the image label and the next object label. Top Right: Overview of existing object labels, Send button. Bottom Right: Address of the current image, interface to reload previous images.*

3.3 Region of Interest Search

Due to the limited computation capabilities of the NAO, not the whole image can be inferenced by a neural network. Therefore, a deterministic region of interest is needed to provide candidate regions to be classified. This is done in multiple steps.

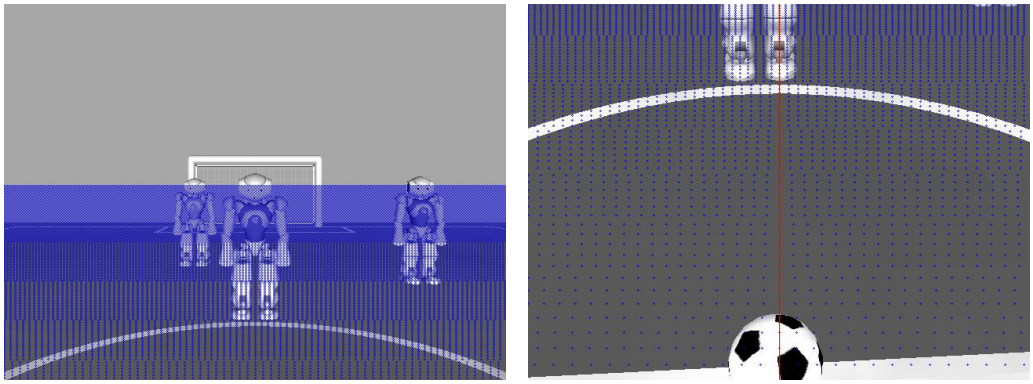
First, the image is preprocessed into gradient based segments. Afterwards, a set of sliding windows is created and scored based on the image segments. The selection of windows to classify is done based on the sliding windows and their scores. Finally, the selected windows positions are corrected and similar ones are merged.

[1] Many thanks to all the HULKs supporting me labeling all those images. Special thanks to the leaders of the annotate high score board: Konrad Nölle, Lasse Peters, Rene Kost, Felix Wege and Pascal Loth.

3.3.1 Image Segmentation

As first step of the region of interest search the image is divided into segments for faster computation. Currently the image segmentation consists of vertical scanlines in every fourth column of the image. With this approach, objects in further distance can disappear between those scanlines while closer area is sampled in a very high resolution. Approaches with projected sample points with fixed distance in world coordinates could be a solution for this problem.

The robots usually have a large backlash in joints and the camera extrinsics are very sensible to forces from outside, e.g. when the robot falls down in a game. Projected approaches suffer from those problems due to the unstable projection. As a compromise the projection is done by a fixed camera to ground matrix which is close to the average projection matrix is used as figure 3.2 illustrates.



(a) *top image* **(b)** *bottom image*

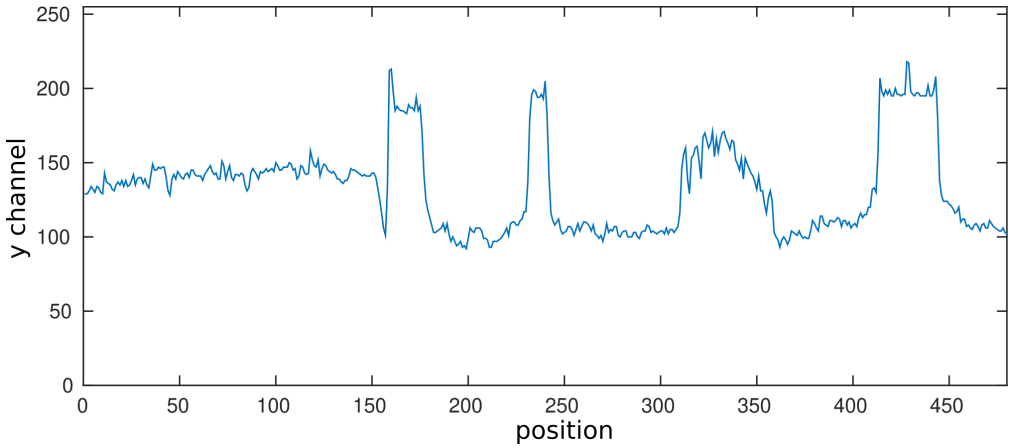
Figure 3.2: *Sample points of horizontal image segmentation.*

The edge detection along the scanlines is done similar to the vertical scanlines [39, pp. 16–18]. The gradient is determined by

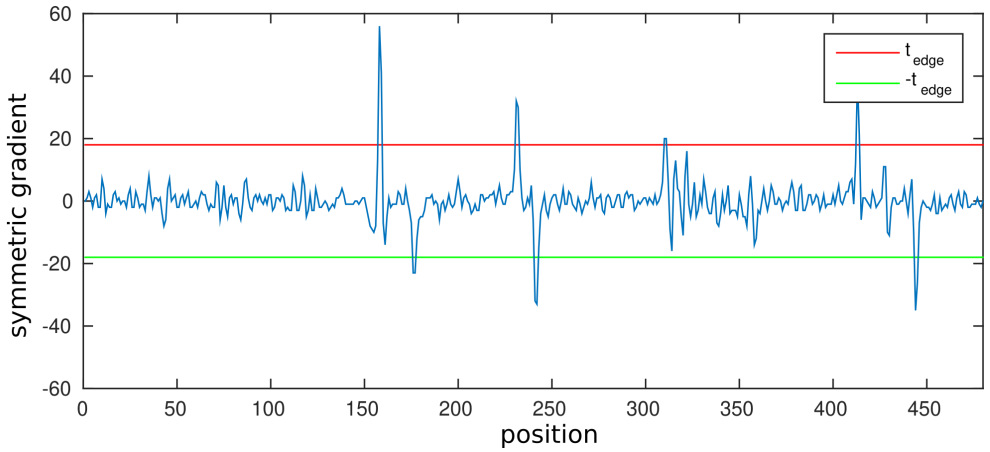
$$g(x) = \frac{1}{2}(f(x-1) + f(x+1)) \tag{3.1}$$

where $f(x)$ corresponds to the image value at pixel x. The local maxima and minima of the symmetrical gradient provide information about the exact position of an edge. The threshold value t_{edge} allows to define a minimum height from which an intensity jump is detected as an edge. The algorithm calculates the symmetrical gradients first between two brightness values. If this is outside the interval $[-t_{edge}, t_{edge}]$ or if it is larger than a temporary maximum gradient, it replaces it. The corresponding position is stored as a

peak. As soon as the next gradient is in the interval $[-t_{edge}, t_{edge}]$, the last stored peak is used as the edge. The symmetric gradient is calculated between two pixels, so this is even in between. The algorithm corrects the peak position xpeak and places the edge at the correct position. Figure 3.3 shows the result of the algorithm for an example scanline.



(a) *Sample signal of the y channel from a scanline with 480 values. [39, p. 16]*



(b) *Visualization of the gradient eq. 3.1 and the interval given by* t_{edge}. *[39, p. 17]*

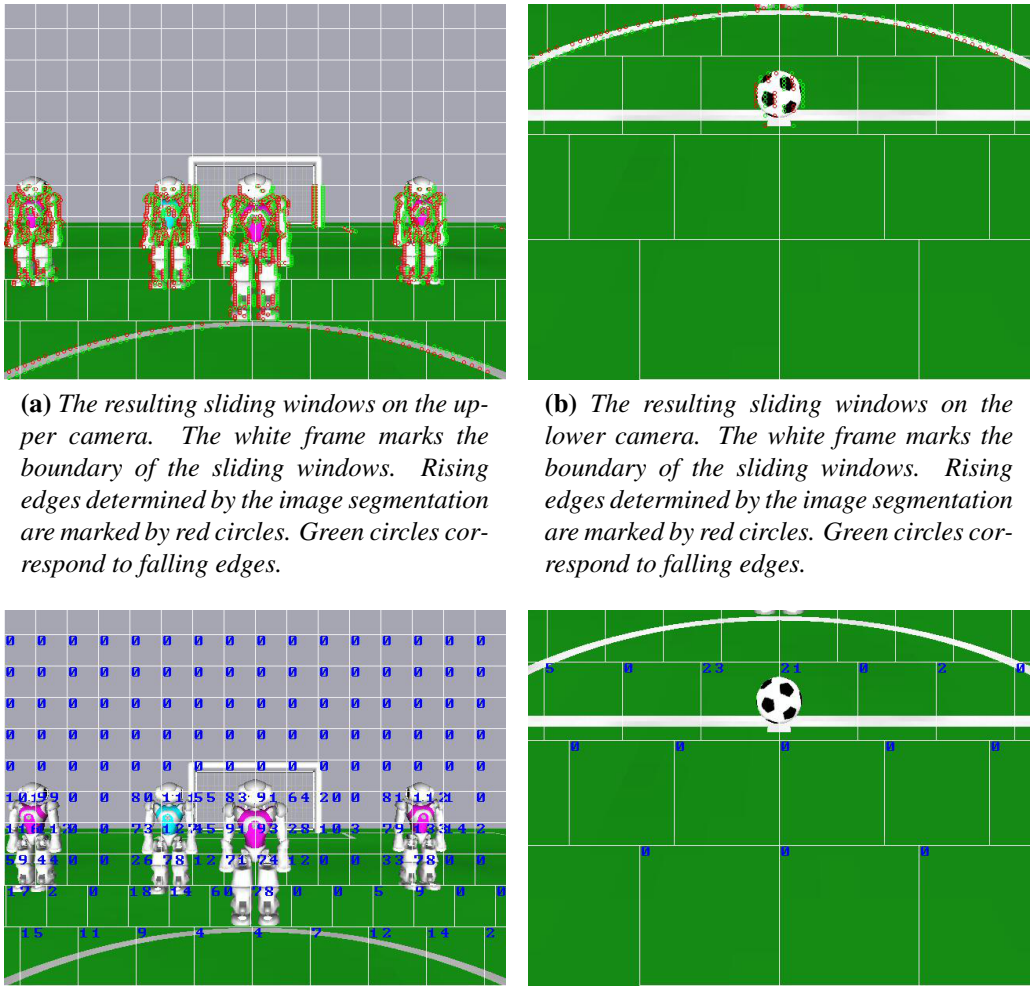
Figure 3.3: *Results of the symmetric gradient algorithm.*

3.3.2 Scored Sliding Windows

The next step is to find areas of the image that may contain objects. In our case, the objects yield much more edges than their background, which mainly consists of plain green carpet. The problem is that robots and balls are of different size and may occlude or overlap each other within the image.

The size issue can be fixed by not searching for the whole robot but just for its feet, which are approximately of the same size as the ball. While one can not prevent occlusion, overlapping becomes less bad when only the lower parts near to the ground are in scope.

In order to find areas which contain many edges, a sliding window of the size of the robots feet is swiped through the image with a step size equal to the window size. For each window, the number of edges based on the horizontal image segments are counted, as figure 3.4 illustrates.



(a) *The resulting sliding windows on the upper camera. The white frame marks the boundary of the sliding windows. Rising edges determined by the image segmentation are marked by red circles. Green circles correspond to falling edges.*

(b) *The resulting sliding windows on the lower camera. The white frame marks the boundary of the sliding windows. Rising edges determined by the image segmentation are marked by red circles. Green circles correspond to falling edges.*

(c) *The resulting scores for the sliding windows on the upper camera.*

(d) *The resulting scores for the sliding windows on the upper camera.*

Figure 3.4: *Results of the scored sliding window module.*

3.3.3 Selection of Sliding Windows

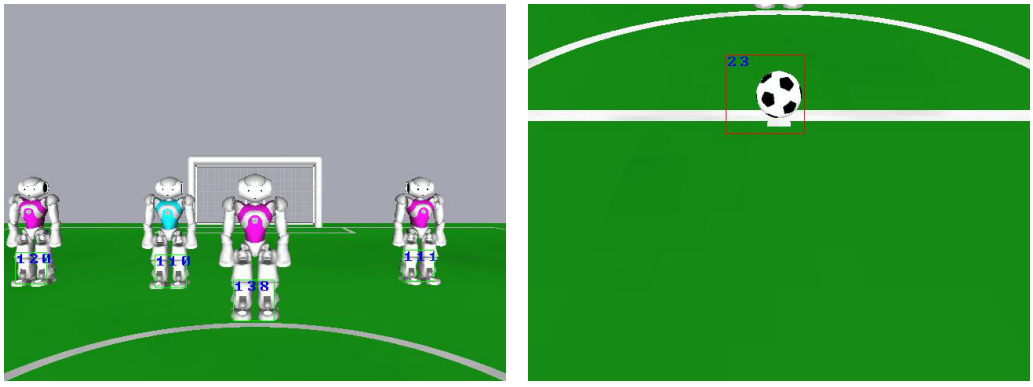
After creating and scoring the sliding windows the windows to classify are selected and corrected in position. A window gets selected for classification when it matches the following conditions. First, the window has to contain more edge points than a configured threshold. Then, the window must be inside the field border determined by the framework. Finally, as a filter for the fields lines, the pearson correlation coefficient for the rising and falling edges is calculated [40, pp. 240–242]:

$$r = \frac{(\sum_{i=0}^{N} x_i y_i) - (\sum_{i=0}^{N} x_i)(\sum_{i=0}^{N} y_i)/N}{\sqrt{(\sum_{i=0}^{N} x_i^2) - (\sum_{i=0}^{N} x_i)^2/N}\sqrt{(\sum_{i=0}^{N} y_i^2) - (\sum_{i=0}^{N} y_i)^2/N}} \tag{3.2}$$

where N is the number of edge points and x_i, y_i their image coordinates. As edge points in windows only containing a field line lay on a line, they result in a low correlation coefficient. Windows with a coefficient lower than a configured threshold won't get selected.

3.3.4 Position Correction and Merging

After selecting windows by the criteria mentioned in the previous section, the windows are still not well aligned on the objects. Therefore, the windows are shifted into the center of their edge points. Afterwards, overlapping windows get merged as algorithm 3 shows. The algorithm also filters areas of windows which are occluded by other windows. Figure 3.5 shows the result of the whole region of interest search.



(a) *top camera* **(b)** *bottom camera*

Figure 3.5: *Resulting regions of interest.*

Algorithm 3 Position correction and merging

```
result = []
occluded = []
for window in selectedWindows do
    if result.empty() then
        result.append(window)
        occluded_from = (window.center.x-window.size.x)
        occluded_to = (window.center.x+window.center+x)
        occluded[occluded_from:occluded_to]=true
        continue
    end if
    merged = false
    for other in result do
        centerDistance = (window.center-other.center).norm()
        maxSize = max(window.size.norm(),other.size.norm())
        overlap = centerDistance / maxSize
        if overlap < overlapThreshold then
            other.center = other.center * other.seeds
            other.center += candidate.center * candidate.seeds
            other.center /= other.seeds + candidate.seeds
            other.size = other.size * other.seeds
            other.size += candidate.size * candidate.seeds
            other.size /= other.seeds + candidate.seeds
            occluded_from=other.center.x-other.size.x
            occluded_to=other.center.x+other.size.x
            occluded[occluded_from:occluded_to] = true
        end if
    end for
    if !merged && !occluded[window.center.x] then
        result.append(window)
        occluded_from = (window.center.x-window.size.x)
        occluded_to = (window.center.x+window.center+x)
        occluded[occluded_from:occluded_to]=true
    end if
end for
```

3.4 Setup of Train and Test Data

The data used for training and evaluation of the classifier was collected during various events:

- RODEO 2018 (Dortmund, Germany)
- Iran Open 2018 (Teheran, Iran)
- German Open 2018 (Magdeburg, Germany)
- Various Test Games (HULKs Laboratories)

The collected data contained 20939 images which were labeled as 'game' meaning that the scene on those images may appear within a real game. Within those images 25467 robots and 5888 balls were annotated. The candidate generation detected 14253 robots and 2637 balls, resulting in a detection rate of 56% for robots and 44.8% for balls. Beside the candidates laying on labeled objects the region of interest search produced 13029 candidates containing no objects.

During training, every candidate gets augmented multiple times with different options. First, the window will be moved by 2 pixels in each direction, producing 9 different samples. Afterwards, the candidate is as is and mirrored along the y-axis, producing 2 different samples for each moved window. With this approach, each candidate produces up to 18 different samples. To balance the training set, the augmentation factor for each class is multiplied by the amount of candidates of this class relative to largest amount of candidates.

Chapter 4

Model Training and Optimization

Using the data generated and labeled using the region of interest search and the label data from annotate, networks can be trained and optimized.

4.1 General Structure of CNNs

The uniform structure used for all evaluated individuals is given in figure 4.1. The input is a YCbCr candidate image of arbitrary size. It is resized to a fixed size using nearest neighbor interpolation. Then, multiple convolutional layers are applied. Finally, multiple fully connected layers are applied. The output is a vector representing the class scores.

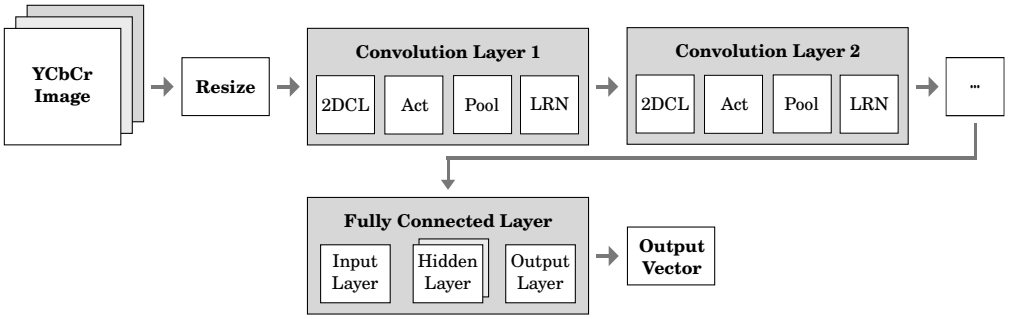


Figure 4.1: *The general structure of a CNN used in the experiments. First, the image is resized to a fixed size. Then, for each convolutional layer a two dimensional convolution mask is applied (2DCL) followed by an activation (Act) and pooling (Pool) layer. After each convolution, a local response normalization is applied. Finaly, the output of the last convolutional layer is fed into a fully connected network which outputs the final output vector.*

Each individual specifies the remaining hyperparameters within this structure. These are the input size, number of convolutional and fully connected layers as well as their internal configuration. Each convolutional layer is parameterized by a mask size, pooling type and activation function. Likewise, the parameters of a fully connected layer consist of the size and activation function. The parameters of this structure yields the search space for the optimization step.

4.2 Search Space

The configuration of a single CNN based on section 4.1 gives the set of hyperparameters to be optimized. The input image is resized within the range $[8,24] \in \mathbb{N}$ for x and y direction. The amount of convolutional layers is limited to four. For each layer, there are five layers at maximum with sizes within $[1,4] \in \mathbb{N}$ in both dimensions. The pooling functions was selected from no pooling, Max-pooling or Avg-pooling. The activation function was either TANH or ReLU, same holds for the fully connected part. There were four hidden layers at maximum in the fully connected part, each with a number of neurons within $[2,20] \in \mathbb{N}$.

4.3 Fitness Function

The fitness function derived from [8] consists of two major parts. The approximation of the inference complexity of a network and the classification performance of a network.

4.3.1 Classification Performance

The classification performance metric described in [8, p. 23] cannot directly be applied to multi-class decision problems. For each network, a k-fold cross validation will be executed, yielding k values for each class c of TNR_{nck}, TPR_{nck} of that network n. In order to approximate a lower bound of the metrics for each class, the difference of the mean and the variance is calculated. To reduce the metrics over the classes, the mean of the result for every class is calculated except for the rejection class. Latter one is omitted as it is linear dependent to the other values. The TPR_n of a network n is computed by:

$$TPR_n = mean_{c_i \in C \setminus c_{\text{reject}}}(mean_{k_i \in [1,k]}(TPR_{nc_ik_i}) - var_{k_i \in [1,k]}(TPR_{nc_ik_i})) \tag{4.1}$$

, where *mean* is the arithmetic mean and *var* is the variance of their arguments.

The TNR_n of a network is calculated analog to the TPR_n:

$$TNR_n = mean_{c_i \in C \setminus c_{\text{reject}}}(mean_{k_i \in [1,k]}(TNR_{nc_ik_i}) - var_{k_i \in [1,k]}(TNR_{nc_ik_i})) \tag{4.2}$$

4.3.2 Inference Complexity

The complexity of a network is asymptotically approximated and linearly scaled. The complexity cc_{ni} of a convolutional layer i of network n is approximated by

$$cc_{ni} = I_x \cdot I_y \cdot I_c \cdot m_x \cdot m_y \cdot m_c \tag{4.3}$$

where I_x, I_y, I_c corresponds to the layer input size and depth, m_x, m_y, m_c to the amount and size of the convolution masks in this layer. The complexity cf_n of the fully connected part of network n is approximated by

$$cf_n = \sum_{i=1}^{k} s_i \cdot s_{i-1} \tag{4.4}$$

where k is the number of hidden layers and s_i the size of layer i. s_0 is the input vector size.

The complexity of a network topology with k convolutional layers is then

$$c_n = \sum_{i=1}^{k} cc_{ni} + cf_n \tag{4.5}$$

For the overall fitness the networks complexity is normalized by the complexity value of the largest possible network $\hat{n}$ within the search space:

$$c_n^* = \frac{c_n}{c_{\hat{n}}} \tag{4.6}$$

4.3.3 Resulting Fitness Function

Given the approximation of the classification performance and the inference complexity, the resulting fitness function is chosen similar to [8, p. 24] as follows.

$$f_n = 0.7 \cdot TNR_n^2 + 0.25 \cdot TPR_n^2 + 0.05 \cdot c_n^* \tag{4.7}$$

As described in section 1.5, the true negative rate is more important than the true positive rate, this component has a way higher weight. As the search space is already limited to topologies which are feasible to inference on the NAO, the inference complexity got a very low weight within the fittness function, in order to prefer smaller networks with similar classification performance.

Chapter 5

Evaluation of Optimization Approaches

This chapter describes evaluation results of approaches for the hyperparameter optimization problem described in chapter 4. First, a Monte Carlo tree search based approach is evaluated in section 5.1. Afterwards an existing genetic approach [12] is evaluated on the new problem and described in section 5.2 The genetic approach is then extended by an elitism variant, as stated in [12, p. 12]. Finally, the results of those approaches are be compared in section 5.3.

To meet the requirements described in section 1.5, the classifier should have a true negative rate above 97.5% with a true positive rate of at least 80% for the ball class. For the robot class, it should achieve a true negative rate above 90% with a true positive rate of at least 80%.

5.1 MCTS Optimization

This section illustrates the application of an MCTS optimization to the framework described in chapter 4 to search of the hyperparameters of a model. Training and Evaluation of the networks generated by the MCTS is done with the data collected in section 3.4.

5.1.1 Setup

In order to optimize the search space of network topologies with MCTS, the structural setup needs to be formulated as board-filling game. The optimized search space is

arranged in a ordered key-value map, as section 5.1.1 shows. First, the input image is sampled within the range $[8,24] \times [8,24]$. The amount of convolutional layers is limited to three. For each layer, there have been five layers at maximum with sizes within $[1,4] \in \mathbb{N}$ in both dimensions. The pooling functions was selected from no pooling, max-pooling or avg-pooling. The activation function was either no activation, TANH or ReLU, latter two for the fully connected part. There were four hidden layers at maximum in the ANN, each with a number of neurons within $[2,20] \in \mathbb{N}$.

```
search_space = {
    "00_sample_size_x": (8,24),
    "00_sample_size_y": (8,24),
    "10_convolution": [
        {
            "00_size_x": (1,4),
            "10_size_y": (1,4),
            "20_out_channels": (1,5),
            "30_pooling": (0,2), # None, Avg, Max
            "40_activation": (0,2) # None, ReLU, TANH
            },
        ]*3,
    "20_fully_connected": {
        "00_layer": [{"size": (2,20)}]*4,
        "10_activation": (1,2) # ReLU, TANH
    }
}
```

Based on this configuration, the order of the nodes in the game tree is defined. A game starts with choosing the sample size in x direction, then in y direction. Afterwards the algorithm may add up to four convolutional layers. For each layer, it needs to choose the number and size of the masks, as well as the pooling and activation function. Then it can add up to 4 hidden layers with a certain size to the fully connected part. After choosing the activation function for the fully connected part, the game is finished.

5.1.2 Results

Figure 5.1 shows the fitness function results of the best network after each iteration. There were 1500 iterations executed, the best network was found in iteration 524. The performance of the algorithm shows a logarithmic behavior over time.

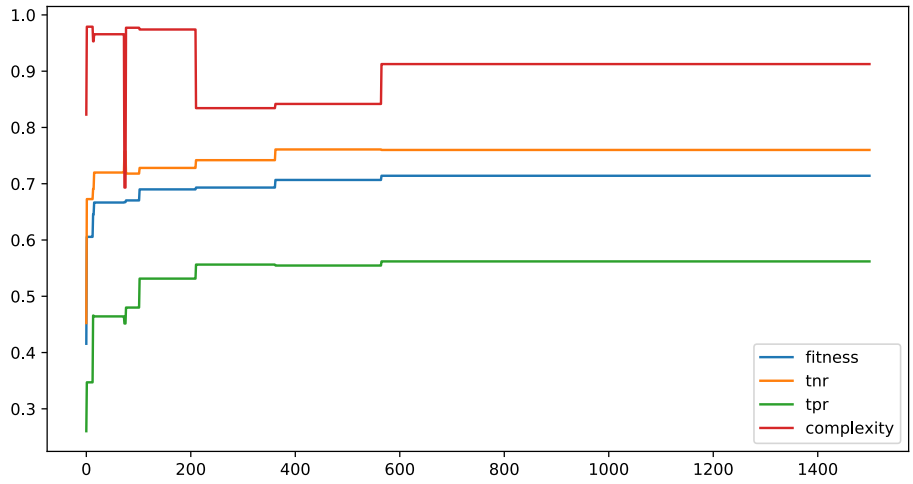


Figure 5.1: *Fitness results of the best network after each iteration. Result of the fitness function is plotted as 'fitness' equation (4.7). 'tnr', 'tpr' refer to the classification performance parts TNR^2 and TPR^2. The model complexity part is shown by 'complexity'*

The detailed progress of the MCTS optimization is listed in table 5.1. The algorithm first tended to use simple structures with no convolutional layers and only small fully connected layers. Then, the chosen model became very complex until iteration 341. Still, the most successful network found by this approach showed to be not at the upper bound of the search space. It used a medium sample size, only a single one dimensional convolutional layer and one fully connected layer.

Table 5.1: *Top 5 Results of the MCTS optimization. Column 'Iteration' refers to the iteration when the network was found. The result of the fitness function is listen in Column 'Score'. 'Sample Size' shows the chosen input size of the network in (x,y). Mask size and amount, Pooling type and Activation Function is shown in Column 'Convolution' Last Column 'Fully Connected' shows the hidden layer sizes as well as the chosen activation function of the fully connected layers. Empty square brackets indicate that there were no hidden layers.*

Iteration	Score	Sample Size	Convolution	Fully Connected
524	0.7141	(17,16)	(4,1), 5, None, ReLU	[19], ReLU
341	0.7067	(22,21)	(3,3), 4, Avg, ReLU (3,2), 5, Max, TANH	[], TANH
200	0.6931	(23,16)	(3,3), 5, Avg, ReLU (2,4), 5, Max, TANH	[14], TANH

Iteration	Score	Sample Size	Convolution	Fully Connected
98	0.6899	(14,23)	(1,2), 4, Max, TANH	[8], TANH
72	0.6703	(12,18)		[13], ReLU

The best network found by the algorithm reached a true negative rate of about 0.8 for the robot and 0.93 for the ball class. Table 5.2 lists the detailed results of this network.

Table 5.2: *Detailed results of the best network found by the MCTS optimization.*

metric	cv	robot	ball	reject
tnr	1	0.77915	0.93768	0.88630
	2	0.80196	0.94378	0.89738
	3	0.83544	0.93439	0.85841
tpr	1	0.74748	0.74375	0.71504
	2	0.77183	0.76195	0.75248
	3	0.70474	0.77072	0.78102

The optimization using MCTS with UCT tree policy was not able to achieve the desired classification results. It slightly missed the requirements of the ball detection with a $TNR \approx 93.9\%$ and $TPR \approx 75.9\%$. For the robot class, it had even a larger difference to the requirements with a $TNR \approx 80.55\%$ and a $TPR \approx 74.13\%$. Still, the results are way better than guessing and there could be a model which solves the problem within the desired error boundaries.

5.2 Genetic Optimization

Previous work on single class problems showed promising results on genetic approaches [12]. In order to compare the novel MCTS approach to current work, the genetic approach will be applied to the new problem described in chapter 4. Results are described in section 5.2.2.

As stated in [12, p. 12], the genetic algorithm could be enhanced using a elitism. Therefore, beside the application of the vanilla version an evaluation of the extended approach is done in section 5.2.3.

5.2.1 Setup

As with the MCTS approach, data, general network structure, search space and fitness function described in chapter 4 are used for the experiments. The parameters for the genetic algorithm are chosen empirical by the conditions mentioned. 30 generations with 50 networks in each generation are evaluated. The worst 10% in each generation are excluded from reproduction. The mutation probability was set to $\frac{1}{22}$ according to the maximum number of degrees of freedom of the given search space.

5.2.2 Results without Elitism

This sections shows the results of the vanilla version of the genetic algorithm described in [12] used with the settings described in section 5.2.1. The best network was found in generation 9 with a fitness of 0.7429.

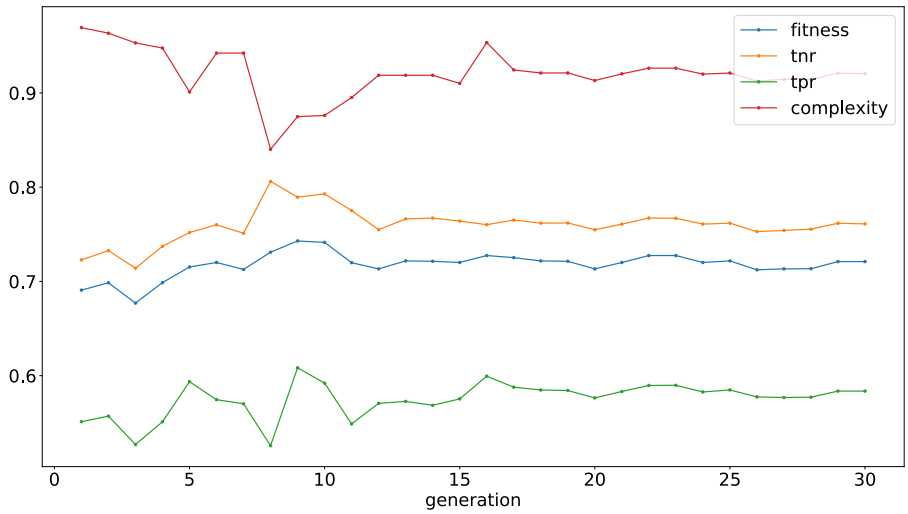


Figure 5.2: *Fitness results of the best network after each generation. Result of the fitness function is plotted as 'fitness' equation (4.7). 'tnr', 'tpr' refer to the classification performance parts TNR^2 and TPR^2. The model complexity part is shown by 'complexity'*

The algorithm showed high improvements in the early generations, the first generation yielded a network with a fitness of 0.6907 which was increased to one with a fitness of 0.7429 in generation 9. However, in the 10th generation a very simple model dominated

the reproduction, pushing the evolution towards a solution with a lower classification performance. Due to the lack of elitism, the algorithm was not able to keep further track of the solution from generation 9 and did not achieve such a fitness again. Figure 5.2 illustrates the fitness of the best networks after each generation.

Table 5.3: *Top 5 Results of the genetic optimization. Column 'Gen' refers to the generation when the network was found. The result of the fitness function is listen in Column 'Score'. 'Sample Size' shows the chosen input size of the network in (x,y). Mask size and amount, Pooling type and Activation Function is shown in Column 'Convolution' Last Column 'Fully Connected' shows the hidden layer sizes as well as the chosen activation function of the fully connected layers. Empty square brackets indicate that there were no hidden layers.*

Gen	Score	Sample Size	Convolution	Fully Connected
9	0.7429	(12,23)	(1,4), 5, ReLU, None (3,2), 5, ReLU, Max (1,3), 5, ReLU, None	[] TANH
10	0.7414	(19,12)	(3,2), 5, ReLU, None (3,2), 5, TANH, Max (2,3), 5, ReLU, None	[] ReLU
8	0.7310	(21,19)	(1,2), 5, ReLU, None (3,2), 5, TANH, Max (1,3), 5, ReLU, None	[] TANH
16	0.7274	(19,19)	(2,1), 5, TANH, Max	[19,10,18] ReLU
29	0.7210	(16,14)	(2,1), 5, TANH, Max	[17, 10, 18], ReLU

The detailed results of the top five networks of the genetic optimization is listed in table 5.3. Networks with three convolutional layers dominated in the most generations. Those networks had none or only one small fully connected layer. This may be due to the reduction of the image sizes by the convolution padding and pooling steps. After the performance drop in the 11th generation the network got stuck at a local maximum of networks with a single convolutional layer and three hidden layers.

Table 5.4: *Detailed results of the best network of the genetic approach.*

metric	cv	robot	ball	reject
tnr	1	0.8115	0.9584	0.8709
	2	0.8220	0.9524	0.8956
	3	0.8325	0.9540	0.8735
tpr	1	0.7555	0.7641	0.7623

metric	cv	robot	ball	reject
	2	0.7884	0.8063	0.7455
	3	0.7650	0.8019	0.7532

Table 5.4 lists the detailed results of this network. The optimization using the genetic algorithm was still not able to achieve the desired classification results. Although it was closer than the MCTS approach it missed the requirements of the ball detection with a $TNR \approx 95.5\%$ and $TPR \approx 79.1\%$. For the robot class, it had even a larger difference to the requirements with a $TNR \approx 82.2\%$ and a $TPR \approx 76.9\%$.

5.2.3 Results with Elitism

The vanilla approach evaluated in section 5.2.2 shows a lack in remembering good decisions of previous generations. This problem can be faced by extending the algorithm with an elitist selection. The results of the genetic algorithm using elitist selection is described in this section. Additionally to the setup chosen in section 5.2.2, the best three organisms are carried over from the current to the next generation, unaltered. The fitness results of those organisms still may have small differences due to the random factors in the training face, e.g. weight initialization or the order of the examples trained.

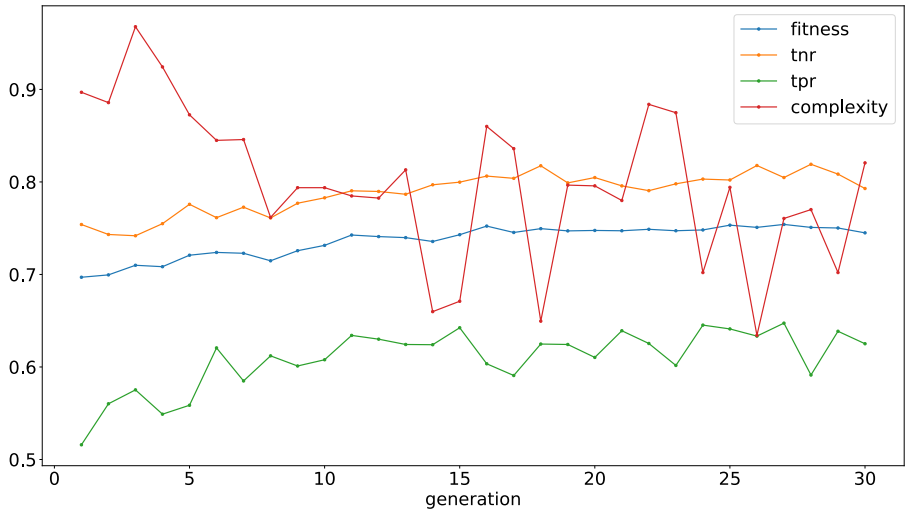


Figure 5.3: *Fitness results of the best network after each generation. Result of the fitness function is plottet as 'fitness' equation (4.7). 'tnr', 'tpr' refer to the classification performance parts* TNR^2 *and* TPR^2*. The model complexity part is shown by 'complexity'*

The winner of the first generation already showed a quite high performance with a fitness of 0.6969. While the classification performance was continuously growing, the models became more complex over time. The elitism approach redressed the problem of forgetting good solutions. In generation 14 a quite complex model dominated even though the fitness was slightly lower due to the higher inference complexity. Two generations later the simpler model took the lead again as figure 5.3 illustrates. Without elitism, a similar effect happened in generation 15 to 18. The vanilla approach was not able to recover as it forgot the more complex model in generation 16. Using elitist selection, the complex model remained in the population and took over several times yielding the best results.

Table 5.5: *Top 5 Results of the genetic optimization with elitism. Column 'Gen' refers to the generation when the network was found. The result of the fitness function is listen in Column 'Score'. 'Sample Size' shows the chosen input size of the network in (x,y). Mask size and amount, Pooling type and Activation Function is shown in Column 'Convolution' Last Column 'Fully Connected' shows the hidden layer sizes as well as the chosen activation function of the fully connected layers. Empty square brackets indicate that there were no hidden layers.*

Gen	Score	Sample Size	Convolution	Fully Connected
27	0.7541	(22,13)	(4,2), 5, ReLU, None (4,3), 5, ReLU, Max	[19] TANH
25	0.7533	(15,17)	(4,2), 5, ReLU, None (4,3), 5, TANH, Max	[19], TANH
16	0.7523	(19,16)	(4,1), 4, ReLU, None (3,3), 5, ReLU, Max	[14], TANH
26	0.7509	(22,17)	(4,3), 5, ReLU, None (4,3), 5, ReLU, Max	[19], TANH
28	0.7509	(22,13)	(4,3), 5, ReLU, None (4,2), 5, ReLU, Max	[14] TANH

The detailed results of the top five networks of the extended genetic optimization is listed in table 5.5. Networks with two convolutional layers dominated all top five generations. Those networks had one fully connected layer with mostly 19 neurons.

Table 5.6: *Detailed results of the best network of the genetic approach using elitsm.*

metric	cv	robot	ball	reject
tnr	1	0.8160	0.9442	0.9157
	2	0.8527	0.9625	0.8858
	3	0.8660	0.9423	0.8729
tpr	1	0.7999	0.8460	0.7060
	2	0.7840	0.8229	0.7952
	3	0.7431	0.8334	0.7862

Table 5.6 lists the detailed results of the best network found. The results for the ball class are very similar to the ones without elitism. For the robot class, the achieved performance was much closer to the desired classification results. Although the extended genetic approach outperformed the vanilla one, it missed the requirements of the ball detection with a $TNR \approx 95\%$ and $TPR \approx 83.4\%$. For the robot class, the difference to

the requirements were much higher with a $TNR \approx 84.5\%$ and a $TPR \approx 77.6\%$.

5.3 Evaluation

The results of the MCTS based optimization from section 5.1 are compared with those of the Genetic Algorithm described in section 5.2. The vanilla version of the genetic algorithm already outperformed the MCTS approach using UCT. The extension of the genetic approach by elitism outperformed the both others.

Figure 5.4 illustrates the performance difference of the algorithms. The result of the MCTS is increasing monotonic and shows a logarithmic behavior. The first generation of the genetic algorithm seemed to converge against a well solution but got stuck on a less fitted local minimum in the later generations. This problem could be faced by adding elitist selection, resulting in the best solution of all three algorithms.

The MCTS already outputs the first result after the first iteration. The genetic algorithm has to compute a whole generation before it outputs the first result. After the fifth generation the vanilla GA superseded the result of the MCTS with a fitness of 0.7153 compared to 0.7141. It was able to extend its advantage to 0.7429 until the 9th generation, then suffering the problems mentioned. The extended version of the GA was able to find an even better solution in the 27th generation with a fitness of 0.7541.

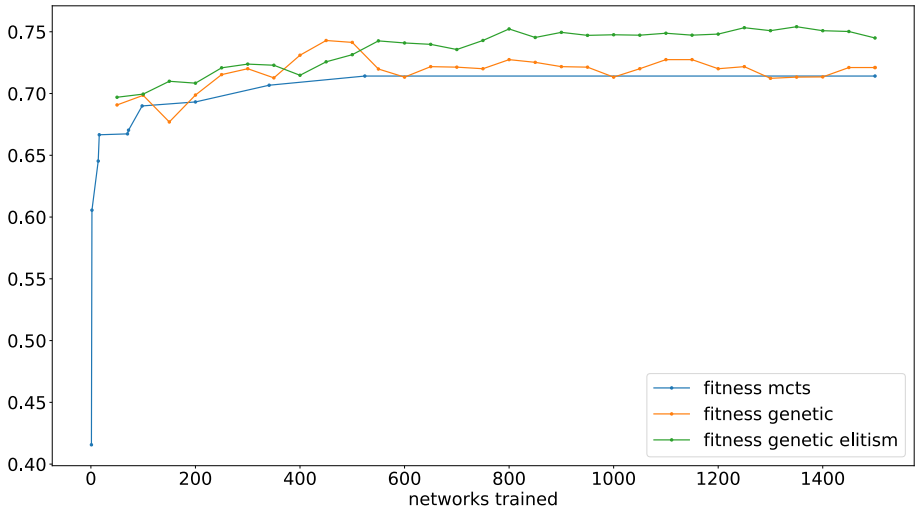


Figure 5.4: *Compared results of the MCTS and Genetic Optimization over the number of trained networks. The genetic algorithm (orange plot) outputs a result after each generation, each generation contains 50 networks. The MCTS algorithm (blue plot) yields the current best result after each model training.*

In the overall result, the genetic algorithm was able to find a solution much closer to the desired requirements described in the previous section. This result could be improved by the extended genetic algorithm. The best network found reached a $TNR \approx 95.5\%$ with a $TPR \approx 79.1\%$ for the ball class and a $TNR \approx 82.2\%$ with a $TPR \approx 76.9\%$ for the robot class.

A drawback of the MCTS using UCT as tree policy is that it is not able to take experience of neighbouring branches within the discovered game tree into account when applying the tree policy. The genetic algorithm implicitly makes use of the experience of all decisions. By changing the MCTS tree policy, e.g. with an SMBO approach, this problem may be resolvable.

Chapter 6

Inference on the NAO

To inference a trained network on the NAO during a game, the network needs to be transferred into the HULKs framework. Therefore serialization of the trained weights and bias variables is needed. Afterwards the HULKs framework is extended by a deserialization capability and a graph inference framework.

After training, the TensorFlow variables can be evaluated to numpy arrays. Those arrays are serialized to JSON.

The graph inference framework is implemented in a object-orientated way. It makes use of the `Eigen/unsupported/Tensor` framework of the eigen library [41]. The graph class takes the path to a JSON file as constructor argument and holds the deserialized JSON object at runtime. When graph is subclassed in order to implement a specific graph, the weights within the JSON object need to be converged to `Eigen::Tensor`. To convert those weights, a `Weight` class is introduced, which takes its dimensions as template parameter as well as a deserialized JSON object containing the weight variable and the variable name as constructor argument. During construction, a weight object automatically converts a JSON list into an `Eigen::TensorFixedSize` according to the given dimensions. A graph implementation may then hold all its weights as member objects. The required graph operations are implemented similar to the eigen backend of TensorFlow [42].

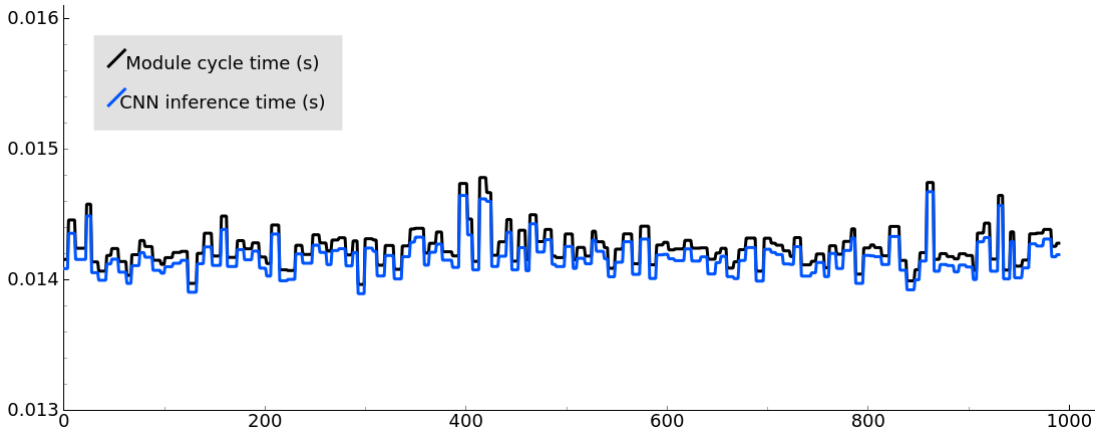


Figure 6.1: *Inference runtime of 1000 cycles on the NAO. The black signal corresponds to the complete cycle time of the object detection module in seconds, including the region of interest search. The inference runtime in seconds is plotted in blue. During the measurement, two candidates where within the image.*

Figure 6.1 illustrates the runtime of the inference framework. During the measurement, two candidates where within the image. The major part of the modules runtime is taken by the CNN inference, as expected. The classification of a single candidate takes about 7.5ms.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

The aim of this thesis was to develop a multi-class object detector to perceive balls and robots at the same time on the NAO robotic system. While implementing a framework for collecting and labeling data, training an optimized CNN for classification, comparing the optimization approaches of MCTS and GAs with and without elitism. Finally, a framework to inference the resulting network on the NAO was developed and evaluated.

The framework for collection and labeling data was built in a way that it can be reused for object detection in general. The collected data contains the whole camera image alongside with the metadata such as the joint angles and sensor data. The objects were annotated directly on the saved images using bounding boxes. Based on this data a region of interest search was introduced producing candidates alongside with ground truth labels that can be used to train a CNN classifier.

To find an optimal classifier, multiple optimization approaches where evaluated, namely MCTS, GA and GA with elitism. While the UCT based MCTS approach was able to find better networks than the random chosen ones in the beginning, the results were not able to met our requirements. The approach was already outperformed by the genetic approach of our previous work [12] but was not able to keep good solutions therefore did not converge against the best solution. The extension of the genetic approach using elitist selection successfully redressed this problem and produced even a better result than the vanilla one.

The resulting classifier has a slightly lower classification performance on the ball class as the current one [8, p. 35]. Additionally to the one-class problem, it is able to detect robots with a classification performance of $TNR \approx 84.5\%$ and $TPR \approx 77.6\%$. The overall detection rate including the candidate generation reached 37.4% for balls and

43.4% for robots. Inference performance of the resulting classifier on the NAO fulfilled our real-time requirements with approximately 7.5ms per candidate.

7.2 Outlook

A problem with MCTS using UCT is that decisions of neighbouring branches in the evaluated tree cannot be taken into account. In future work, this problem could be overcome using different tree policies like SMBO with LSTM based tree policies.

Given a object detection framework, the labeling tool could be extended. Images which are to be labeled could be evaluated by the framework first, yielding label proposals. The users of the labeling tool would then only have to recheck and correct these.

The recently released version of the NAO robot [43] has more computation power with a quad core CPU with integrated GPU. A promising approach for future object detection could be models like faster rCNN [44] which would make a heuristic based region of interest search obsolete.

References

[1] C.-J. Tan, M. Campbell, F.-h. Hsu, J. H. Jr., J. Brody, and J. Benjamin, "Deep blue - overview." [Online]. Available: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[2] J. P. Laboratory, "Mars pathfinder mission," 30-May-2018. [Online]. Available: https://marsprogram.jpl.nasa.gov/MPF/index0.html

[3] R. Federation, "A brief history of robocup," 2016. [Online]. Available: http://robocup.org/a_brief_history_of_robocup

[4] S. R. Europe, "Who is nao?" 2017. [Online]. Available: https://www.ald.softbankrobotics.com/en/robots/nao

[5] S. R. Europe, "NAO humanoid robot platform datasheet," 2017. [Online]. Available: https://www.ald.softbankrobotics.com/sites/aldebaran/files/datasheet_nao_next_gen_en.pdf

[6] R. T. Committee, "RoboCup standard platform league (nao) rule book: 2018 rules, as of may 14, 2018," May 2018 [Online]. Available: http://spl.robocup.org/wp-content/uploads/2018/04/SPL-Rules_small.pdf

[7] R. Federation, "RoboCupSoccer - standard platform," 2016. [Online]. Available: http://robocup.org/leagues/5

[8] G. C. Felbinger, "A genetic approach to design convolutional neural networks for the purpose of a ball detection on the nao robotic system," Project Work, 2017 [Online]. Available: https://www.hulks.de/_files/PA_Georg-Felbinger.pdf

[9] C. Kahlefendt, "A comparison and evaluation of neural network-based classification approaches for the purpose of a robot detection on the nao robotic system," Project Work, 2017 [Online]. Available: http://www.hulks.de/_files/PA_Chris-Kahlefendt.pdf

[10] J. Menashe *et al.*, "Fast and precise black and white ball detection for robocup soccer," in *Robot world cup*, 2017.

[11] Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks for image classification," *CoRR*, vol. abs/1710.10741, 2017 [Online]. Available: http://arxiv.org/abs/1710.10741

[12] G. C. Felbinger, P. Götsch, P. Loth, L. Peters, and F. Wege, "Designing convolutional neural networks using a genetic approach for ball detection," in *Proc. RoboCup 2018 symposium*, 2018.

[13] C. Liu *et al.*, "Progressive neural architecture search," *CoRR*, vol. abs/1712.00559, 2017 [Online]. Available: http://arxiv.org/abs/1712.00559

[14] R. Negrinho and G. Gordon, "DeepArchitect: Automatically Designing and Training Deep Architectures," *ArXiv e-prints*, Apr. 2017.

[15] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proceedings of the 5th international conference on learning and intelligent optimization*, 2011, pp. 507–523 [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25566-3_40

[16] N.-T. HTWK, "Team research report 2017," 2017 [Online]. Available: http://robocup.imn.htwk-leipzig.de/documents/TRR_2017.pdf

[17] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," 1995.

[18] F. Flöck, 2016 [Online]. Available: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#/media/File:K-fold_cross_validation_EN.jpg

[19] G. Developers, "About tensorflow," 15-Aug-2017. [Online]. Available: https://www.tensorflow.org/#about-tensorflow

[20] Glosser.ca, *Artificial neural network with layer coloring*. 2003 [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg

[21] W. Purves, D. Sadava, G. Orians, and H. Heller, *Biologie*. Elsevier Spektrum Akademiker Verlag, 2004.

[22] Y. LeCun, C. Cortes, and C. J. Burges, "THE mnist database of handwritten digits," 2017. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[23] R. Benenson, "Classification datasets results," 22-Feb-2016. [Online]. Available: https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[24] N. O. Lüders, "Object localization on the nao robotic system using a deep convolutional neural network and an image contrast based approach," 2016 [Online]. Available: https://www.hulks.de/_files/MA_Olaf_Lueders.pdf

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in neural information processing systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105 [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[26] G. Developers, 16-May-2018. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/nn/local_response_normalization

[27] C. B. Browne *et al.*, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[28] S. Harbich, "Einführung genetischer algorithmen mit anwendungsbeispiel," Dec. 2007.

[29] S. Baluja and R. Caruana, "Removing the genetics from the standard genetic algorithm," in *International conference on machine learning (icml)*, 1995 [Online]. Available: http://www.ri.cmu.edu/pub_files/pub2/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf

[30] G. C. Felbinger, "A web-based image annotation approach." 09-May-2018 [Online]. Available: https://gitlab.com/gfelbing/annotate

[31] Wikipedia, "Representational state transfer — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Representational%20state%20transfer&oldid=840002995, 2018.

[32] Wikipedia, "Application programming interface — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Application%20programming%20interface&oldid=839782886, 2018.

[33] Wikipedia, "TypeScript — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=TypeScript&oldid=838415153, 2018.

[34] Wikipedia, "OAuth — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=OAuth&oldid=833649422, 2018.

[35] M. Zabriskie, "Promise based http client for the browser and node.js." 09-May-2018 [Online]. Available: https://github.com/axios/axios

[36] I. Facebook, "A declarative, efficient, and flexible javascript library for building user interfaces." 09-May-2018 [Online]. Available: https://github.com/facebook/react

[37] T. B. Authors, "The most popular html, css, and javascript framework for developing responsive, mobile first projects on the web." 09-May-2018 [Online]. Available: https://github.com/twbs/bootstrap

[38] D. Abramov, "Predictable state container for javascript apps." 09-May-2018 [Online]. Available: https://github.com/reactjs/redux

[39] P. Loth, "Implementierung und evaluation einer robusten echtzeitkantendetektion auf dem humanoiden nao-robotiksystem," Bachelor Thesis, 2015 [Online]. Available: https://www.hulks.de/_files/BA_Pascal-Loth.pdf

[40] K. Pearson, "Notes on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, no. 58, Jun. 1895.

[41] G. Guennebaud, B. Jacob, and others, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[42] G. Developers, "Computation using data flow graphs for scalable machine learning," 20-May-2018. [Online]. Available: https://github.com/tensorflow/tensorflow

[43] S. R. Europe, "NAO v6 - overview," 2018. [Online]. Available: http://doc.aldebaran.com/2-8/

[44] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015 [Online]. Available: http://arxiv.org/abs/1504.08083

Appendix A

Detailed Experiment Results

A.1 Detailed Results of MCTS

Table A.1: *Top 5 Results of the genetic optimization with elitism. Column 'Iteration' refers to the iteration when the network was found. The result of the fitness function is listen in Column 'Score'. Columns 'TNR', 'TPR' and 'Complexity' show the parts TNR, TPR and c_n of equation (4.7).*

Iteration	Score	TNR	TPR	Complexity
1	0.4156	0.4525	0.2602	0.8229
2	0.6056	0.6727	0.3471	0.9790
14	0.6453	0.6907	0.4657	0.9530
16	0.6666	0.7199	0.4642	0.9656
70	0.6673	0.7577	0.4516	0.6932
72	0.6703	0.7180	0.4799	0.9772
98	0.6899	0.7281	0.5314	0.9741
200	0.6932	0.7419	0.5563	0.8343
341	0.7067	0.7609	0.5546	0.8418
524	0.7142	0.7600	0.5620	0.9126

A.2 Detailed Results of Genetic Algorithm

Table A.2: *Top 5 Results of the genetic optimization with elitism. Column 'Gen' refers to the generation when the network was found. The result of the fitness function is listen in Column 'Score'. Columns 'TNR', 'TPR' and 'Complexity' show the parts TNR, TPR and c_n of equation (4.7).*

Gen	Score	TNR	TPR	Complexity
1	0.6908	0.7229	0.5511	0.9693
2	0.6985	0.7327	0.5570	0.9634
3	0.6770	0.7140	0.5269	0.9531
4	0.6988	0.7373	0.5510	0.9476
5	0.7153	0.7519	0.5936	0.9010
6	0.7201	0.7601	0.5745	0.9423
7	0.7127	0.7511	0.5702	0.9423
8	0.7310	0.8062	0.5257	0.8402
9	0.7429	0.7894	0.6084	0.8749
10	0.7414	0.7929	0.5920	0.8761
11	0.7199	0.7751	0.5488	0.8952
12	0.7133	0.7549	0.5706	0.9188
13	0.7218	0.7663	0.5727	0.9188
14	0.7214	0.7672	0.5686	0.9188
15	0.7201	0.7641	0.5753	0.9101
16	0.7274	0.7602	0.5995	0.9534
17	0.7252	0.7651	0.5878	0.9244
18	0.7218	0.7651	0.5878	0.9244
19	0.7214	0.7651	0.5878	0.9244
20	0.7133	0.7651	0.5878	0.9244
21	0.7201	0.7651	0.5878	0.9244
22	0.7274	0.7651	0.5878	0.9244
23	0.7274	0.7651	0.5878	0.9244
24	0.7201	0.7651	0.5878	0.9244
25	0.7218	0.7651	0.5878	0.9244
26	0.7123	0.7651	0.5878	0.9244
27	0.7133	0.7651	0.5878	0.9244
28	0.7135	0.7651	0.5878	0.9244
29	0.7210	0.7651	0.5878	0.9244
30	0.7210	0.7651	0.5878	0.9244

A.3 Detailed Results of Genetic Algorithm using Elitism

Table A.3: *Top 5 Results of the genetic optimization with elitism. Column 'Gen' refers to the generation when the network was found. The result of the fitness function is listen in Column 'Score'. Columns 'TNR', 'TPR' and 'Complexity' show the parts TNR, TPR and c_n of equation (4.7).*

Gen	Score	TNR	TPR	Complexity
1	0.6970	0.7540	0.5159	0.8969
2	0.6995	0.7432	0.5602	0.8857
3	0.7099	0.7418	0.5753	0.9678
4	0.7084	0.7549	0.5490	0.9243
5	0.7208	0.7758	0.5586	0.8725
6	0.7238	0.7614	0.6206	0.8449
7	0.7229	0.7727	0.5849	0.8458
8	0.7148	0.7611	0.6120	0.7617
9	0.7257	0.7770	0.6011	0.7938
10	0.7314	0.7828	0.6078	0.7938
11	0.7426	0.7904	0.6341	0.7849
12	0.7409	0.7897	0.6301	0.7826
13	0.7398	0.7867	0.6243	0.8131
14	0.7356	0.7969	0.6241	0.6598
15	0.7430	0.7998	0.6424	0.6710
16	0.7523	0.8063	0.6035	0.8600
17	0.7454	0.8039	0.5908	0.8360
18	0.7495	0.8175	0.6248	0.6495
19	0.7471	0.7990	0.6243	0.7966
20	0.7476	0.8047	0.6103	0.7958
21	0.7472	0.7957	0.6391	0.7800
22	0.7488	0.7906	0.6254	0.8839
23	0.7472	0.7980	0.6017	0.8748
24	0.7481	0.8031	0.6453	0.7021
25	0.7533	0.8021	0.6411	0.7942
26	0.7509	0.8178	0.6334	0.6342
27	0.7541	0.8047	0.6474	0.7605
28	0.7509	0.8191	0.5913	0.7701
29	0.7502	0.8085	0.6386	0.7021
30	0.7450	0.7929	0.6251	0.8206