Technische Universität Hamburg-Harburg
Vision Systems

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

A genetic approach to design convolutional neural networks for the purpose of a ball detection on the NAO robotic system

Project Thesis

Georg Christian Felbinger

04.10.2017

Author's Declaration

I solemnly declare that I have written this thesis independently, and that I have not made use of any aid other than those acknowledged in this thesis. Neither this research paper, nor any other similar work, has been previously submitted to any examination board.

Hamburg, 04.10.2017 Georg Christian Felbinger

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is written in the context of the RoboCup Team HULKs, which is participating in the Standard Platform League (SPL). This chapter provides brief information 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

The RoboCup was founded in 1997, after the success of the chess computer IBM Deep Blue against Garri Kasparow. The idea was to build soccer robots, which are able to beat the human World Cup champion team by 2050. Today, the RoboCup consists of various soccer leagues, such as competitions for housekeeping, rescue or industrial robots. [7]

1.2 The NAO Robotic System

The NAO Robotic System, thereafter 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 [6], an example can be seen in 1.1. The NAO is used by the RoboCup SPL Teams as well as in this thesis.

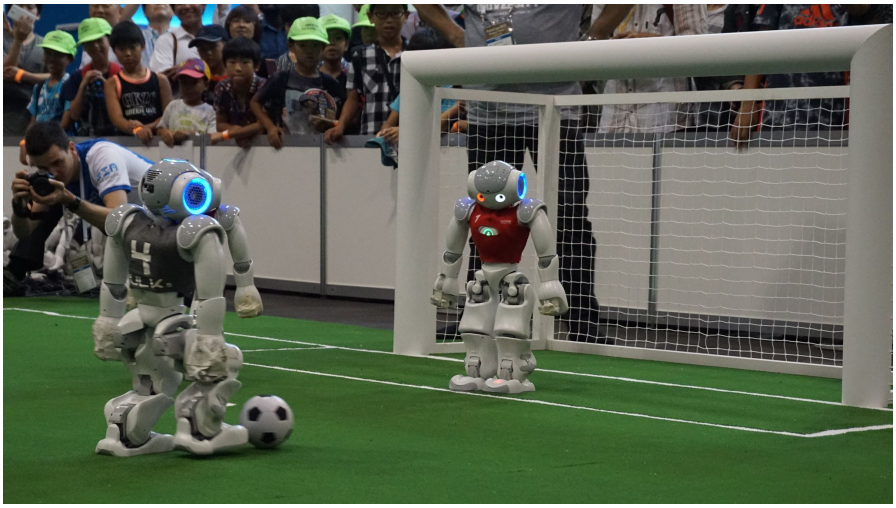

Figure 1.1: *HULKs NAO robot before scoring a goal against UChile at the RoboCup 2017 in Nagoya.*

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. Within the SPL, all participating teams use the same robot, namely the NAO robot. As every team uses this platform, this league is focusing on implementing fast and robust software for real-time soccer purposes.

The games are usually played five versus five. These robots play fully autonomously and each one takes decisions separately from the others, but they still have to play as a team by using communications. Two teams play on a green field with white lines and goal posts, with no other landmarks. 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. [8]

1.4 Motivation

Until 2015, a plain red street hockey ball was used for the games, which could be easily detected using classic image processing algorithms. Since 2016, a black-white patched ball was introduced into the SPL. The previous solution in the HULKs framework was derived from the red ball detection, suffering from many false positives and low detection ranges.

Approaches based on convolutional neural networks for object detection lead to promising results in our previous work, such as the robot detection [14]. These methods save a lot of work, as no manual feature extraction is nessecary. But there are still many hyperparameters for the structural setup of the networks. The central idea of this thesis is to determine these parameters by a genetic optimization approach.

1.5 Overview of existing Approaches

This section briefly explains solutions of other teams ball detections. The common approach for object detection in all teams consists of two steps. First, candidate regions which may contain the object must be found. Second, found candidate regions are classified. The work of the following teams will be explained: B-Human, Nao-Team HTWK and Bembelbots.

1.5.1 Nao-Team HTWK

The candidate generation of HTWK scans the image for blocks below a pre-calculated field limitation with high contrast characteristics. The difference of the CB channels between the inner and outer regions of the object are calculated by the means of the estimated ball size afterwards. Black and white colored pixels have higher values in the CB channel than green colored (field) pixels.

The classification is separated into two stages to save computing time. The first stage is a two hidden-layer neural network, which classifies about ten hypotheses per image, each of size 9x9 pixels. The second stage is an eight layer CNN, which uses 20x20 pixel patches. It only classifies the hypothesis with the best score given by the first stage. The statistics of their testset are 90.3% recall and 99.3% precision. The detailed description can be found in the team research report from 2016 [12].

1.5.2 B-Human

The approach of B-Human does not depend on a trained model at all [2].

The candidate generation scans the image vertically using scan lines of different density based on the size of the ball. To determine ball candidates, these scan lines are searched for sufficiently large gaps in the green that also have a sufficiently large horizontal extension and contain enough white. The candidates position is afterwards improved by evaluation of the contour in a contrast-normalized Sobel image of the candidate.

The classification step is done by multiple filters:

1. Thresholding the response of the contour evaluation in the previous step
2. Comparision of the estimated radius with the projected radius
3. Dropping candidates inside detected robots which are not in the lower area
4. Checking the surface pattern of candidates, which are not surrounded by field color

1.5.3 Bembelbots

The candidates for the training phase were generated from a simulated environment [11, pp. 2-5]. The trained model was tested against real data, yielding a accurasy of approximately 91% [11, pp. 9].

1.6 Goals

The aim of this thesis is to optimize the structure of a convolutional neural network (CNN) using a genetic algorithm. The resulting network should be inferencable on the NAO during a game. The runtime per candidate should be higher than 25 ms on the NAO. During such a game, the measurements of the ball detection are used to maintain a world model over time. A ball which is not detected in an image will still be within the world model. On the other side, a detected ball which is actually not there can easily lead to a wrong world model. Hence for the classification performance, the true negative rate is more important than the true positive rate. The resulting network should have a true negative rate above 95 % with a true positive rate of at least 80 % to be usefull within a SPL game.

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 aquisition chapter shows, how the data for learning the models was generated and set up.

After the data is set up properly, the network architecture can be optimized for the problem using a genetic algorithm, as described in chapter genetic desgin of CNNs. The according experiments and their results can be found in chapter experiments and evaluation. Further evaluations like an input optimization and generalization test of the resulting network is described in chapter further evaluation.

Finally, chapter conclusion and outlook summarizes the thesis and give 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 genetic algorithms, 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 genetic algorithms 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. The set of positives p is the set of example candidate images containing a ball. The other candidate images are assigned to the set of negatives n.

True positives (tp) are those examples from the setup p, which were assigned to a ball by the classifier. The true negatives (tn) are those examples from the sets n, which were not assigned to a ball, respectively. False positives / false negatives (fp / fn) are the images from p / n, which were assigned to the wrong class, respectively.

The true positive rate is the relative amount of correctly assigned examples within the set p: $tpr = \frac{|tp|}{|p|}$. The true negative rate is analog: $tnr = \frac{|tn|}{|n|}$.

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 exclusive 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 [15, pp. 2-3]. Figure 2.1 illustrates a 4-fold cross validation.

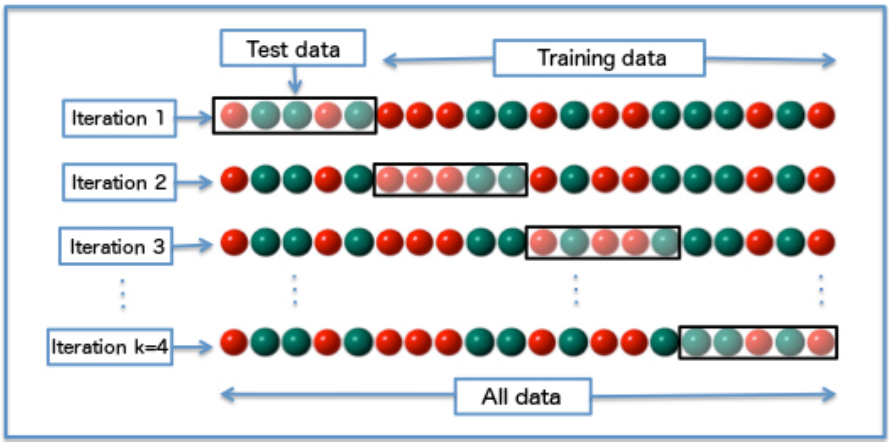

Figure 2.1: *Diagram of k-fold cross-validation with k=4. [9]*

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 debbuging purposes, together with the OFA Tool. Beside the NAO, it has multiple more compile targets, i.e. for running the software with SimRobot.

2.2.2 HULKs OFA

The HULKs OFA tools is a debbuging and configuration tool for the HULKs NAO Framework. It is used to visualize results of vision pipeline such as the image segmenter, ball detection, etc.

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. [3]

2.3 Genetic algorithm

Genetic algorithms (GAs) were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems. It was rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. [19, pp. 3]

There is no specific definition of a genetic algorithm. But most of the proposed algorithms which are called genetic, have common properties: A series of poplations of individuals, a fitness function yielding how well a individual solves the problem, a recombination / mutation function generating new populations based on the fitness values.

The algorithm used in this thesis is based on [10]. The basic elements are chromosomes or individuals $c \in \mathbb{C}^k$, a possible solution in given, k-dimensional search space. Every individual can be assigned to a value which reflects how well it fits to the problem, using a fitness function $f(c) : \mathbb{C}^k \mapsto \mathbb{R}$. A set of n chromosomes which is evaluated in a training step j is called population $P_j = \{c_1, ..., c_n\} \subset S$.

The iterative steps of the algorithms are:

0. Generate random population $P_0 \in \mathbb{C}^k$
1. For every generation j:
    1. Calculation of fitness $F_j = (f(c_1), ..., f(c_n))^T \in \mathbb{R}^n, c_i \in P_{j-1}$
    2. Selection of individual for reproduction $S_j = \text{select}(P_j, F_j)$
    3. Mutation of selected individuals $M_j = \text{mutate}(S_j)$
    4. Recombination of mutated elements, resulting in new generation $P_j = \text{recomb}(M_j)$

2.3.1 Selection

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

$$p(c_i) = \frac{f(c_i) - \text{minscore}}{\sum_{j=1}^{m}\left(f(c_j) - \text{minscore}\right)} \tag{2.1}$$

Hence the indiviual with the lowest fitness value gets is assigned to the surviving probability 0. According to this distribution, n elements are sampled for mutation and reproduction.

2.3.2 Mutation

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

2.3.3 Reproduction

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

2.4 Artificial Neural Networks

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

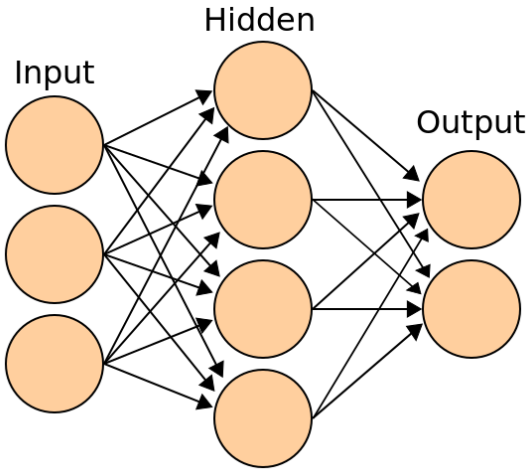

Figure 2.2: *An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another. [@wiki-ann-struct]*

2.4.1 Neurons

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

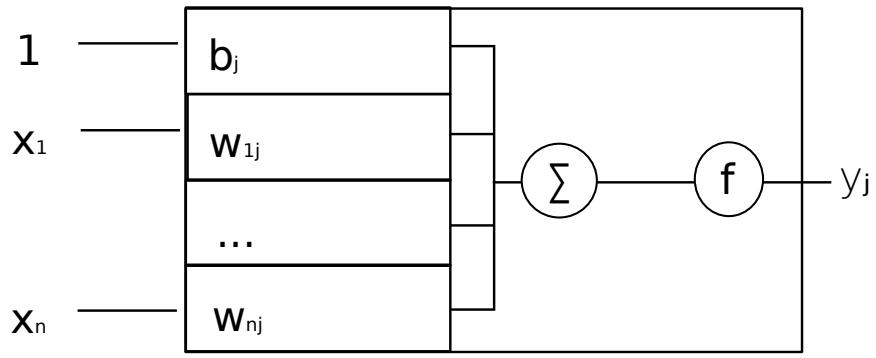

Figure 2.3: *The functioning of a single neuron j. First, the input vector gets element-wise multiplied by the weight vector. Afterwards the bias is added to the sum of the resulting vector. Finally, the value is applied to the activation function, yielding the output of the neuron.*

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 vecotr $b \in \mathbb{R}^k$. The result of neuron j can be computed as $y_j = f(W_{.j}x^T + b_j)$, where the $W_{j.}$ is the j-th column of W. Hence a whole layer of neurons can be computed as $y = f(Wx^T + b)$.

2.4.2 Activation functions

In biology, a neuron "fires" in a non-linear way depending on the input [20, pp. 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.

2.4.2.1 Hyperbolic tangent

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.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

2.4.2.2 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.

$$\text{relu}(x) = \begin{cases} x & ,x < 0 \\ 0 & ,otherwise \end{cases} \tag{2.3}$$

2.5 Convolutional Neural Networks

CNNs have become very popular in the 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 lead to the best results with a test error rate of down to 0.23 [17]. In the CIFAR-10 dataset, CNNs achieved a accuracy of up to 96.53% [1].

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

2.5.1 Convolutional Layer

As the name suggests, 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. 2.4 shows the computation of a convolutional layer.

$$\begin{gathered} y_{i,j,k} = \sum_{di,dj,q} x_{i+di,j+dj,q} \cdot m_{di,dj,q,k} \\ x \in \mathbb{R}^{i \times j \times q}, m \in \mathbb{R}^{di \times dj \times q \times k} \end{gathered} \tag{2.4}$$

2.5.2 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. In this thesis, only $\max(a,b,c,d)$ (maximum value of arguments) and $\text{avg}(a,b,c,d)$ (arithmetic mean of arguments) are used. Figure 2.4 demonstrates a 2x2 max-pooling layer for a single channel image.

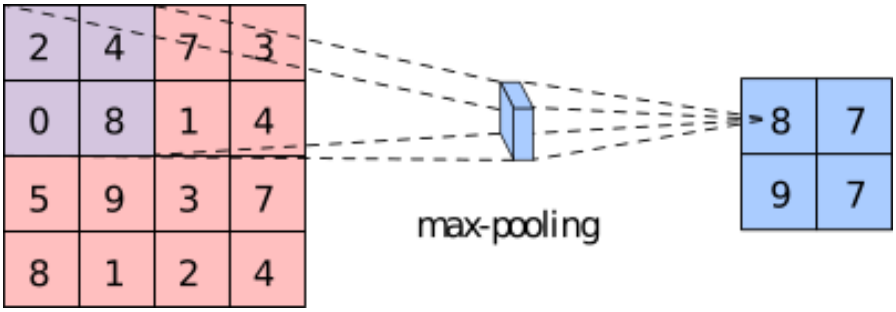

Figure 2.4: *Schematic drawing of a 2x2 sized max-pooling layer downsampling an 4x4 input to 2x2 with a stride of 2 so that no overlapping occurs. [18]*

2.5.3 Normalization layer

Batch Normalization Layers are used to increase the learning rates and to reduce the sensivity to the initialization of the weights [13].

During training, the normalization is calculated batch-wise. For input vectors 1...m:

$$BN_{\gamma,\beta}(x_i) = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B + \varepsilon}} + \beta \tag{2.5}$$

[13, pp. 3].

The scale γ and offset β are trainable parameters which get optimized due to the training problem. The batch mean is elementwise computed by $\mu_B = \frac{1}{m}\sum_{j=1}^{m} x_j$. The batch variance is also elementwise computed by $\sigma_B = \frac{1}{m}\sum_{j=1}^{m}(x_j - \mu_B)$.

For the inference, the mean and the variance are approximated by a moving average approach during the training [13, pp. 4]. The mean μ_n and the variance σ_n after the n-th batch can be computed recursively using the batch mean and variance:

$$\begin{aligned} \mu_n &= \mu_{n-1} \cdot \delta + \mu_{B_n} \cdot (1-\delta) \\ \sigma_n &= \sigma_{n-1} \cdot \delta + \sigma_{B_n} \cdot (1-\delta) \end{aligned} \tag{2.6}$$

Chapter 3

Data Aquisition

This chapter describes the aquistion of the data used for training and testing. It is splitted in the following sections: The algorithm to generate candidates from an image to classify and the labeling procedure candidates.

3.1 Candidate Generation

The previous candidate generation has hard dependencies on the field color detection. From 2017, all games are played on an artificial green turf instead of a green carpet, the field color approach gets worse.

A new approach has been implemented based on the vertical scanline image and projection. After classifying, the generated candidate images, including the result of the current classifier for validation purposes are logged. This ensures that the candidates used for training the network are generated in the same way as during inference. Another advantage is that the result of the classification can directly be saved with the candidate. This enables the possibility to directly evaluate an old classifer during the collecting of new data.

3.1.1 Available Data

This section briefly describes the already available data within the HULKs Framework that can be used.

3.1.1.1 Raw Image

The raw image is available with a resolution of 640x480 pixels. The color space is the YCbCr 422. The brightness given within the y channel is stored for every pixel. The color information given within the cb and cr channels is shared with the neighbouring pixel. The resulting data structure of two consecutive pixels in the image data is therefore `y0 cb y1 cr`

3.1.1.2 Segmented image

Within the HULKs Framework, the raw image is already preprocessed for fast computation. The image is segmented using vertical scanlines on every second column of the 640x480 raw image. Along those scanlines, it computes the two dimensional gradient of the image. Whenever this gradient gets greater than a configured threshold, a new segment along this scanline is created. The color of each segment is determined by a median of five pixels with equal distance along it.

Figure 3.1: *The result of the image segmenter in the HULKs Framework. Top Left: The raw image from the upper camera. Top Right: The raw image from the lower camera. Bottom Left: The segmented image of the upper camera. Bottom Right: The segmented image of the lower camera. Areas above the horizon determined by the projected distance are ignored.*

3.1.1.3 Camera Matrix

The camera matrix contains the information of the intrinsic and extrinsic parameters. It provides an API for convient projection of world and pixel coordinates.

While the intrinsic parameters are calibrated by hand, the extrinsics (camera to ground) are determined during runtime. Given the assumption that at least one of the robots feet stands on the ground, the extrinsic parameters are determined by the joint angles of the robot.

3.1.2 Generating Seeds

As the first step of the candidate generation, the algorithm determines seeds for possible candidates. A seed is the centering pixel of a segment, which passes a series of checks:

1. First, y channel of the corresponding region must be lower than 100 which naturally corresponds to the black patches of the ball.
2. Afterwards, the corresponding ball radius in pixels r_p at the seeds position is calculated using the camera matrix. If the ratio $r_s = \frac{l_s}{r_p}$ of the segments length l_s to the pixel radius is not within the range $[0.1, 0.7]$, the seed is dropped.
3. Afterwards the neighbouring areas of the black patch are checked. Therefore the y values in eight directions around the seed with a distance of $r_p/2.5$ are sampled. All of those sampled values must be at least 8 higher than the y value of the seed. Also, at least 5 of those value must be at least 25 higher than the y value of the seed.

If all conditions match, the seed is used for the candidate generation.

Figure 3.2: *Found seeds on a ball. Blue crosses illustrate the neighbouring checks of seeds. The center of each cross is at the position of a seed. The end points correspond to the sampled pixels for the checks.*

3.1.3 Merging seeds to candidates

After the calculation of all seeds in an image, nearby seeds get merged to candidates. First, an empty set of candidates is initialised.

For every seed in the image, it is checked if there is a nearby candidate with a maximum distance of a balls diameter in pixels. If a candidate is found, the current seed will be merged into that candidate, by taking the mean of the position and radius. If no candidate was found, a new candidate with position and radius of the current seed is added to the set of candidates. The candidates are filtered afterwards, such that only candidates based on at least 2 seeds are left.

Figure 3.3: *Candidate (green circle with green cross in center) derived from merged seeds. The seeds (blue crosses) got merged as they are within the distance of the balls diameter. The candidates center and radius are determined by the mean of the projected radii and centers of the seeds.*

3.1.4 Reprojection of found balls

The measurements of the ball detection are tracked within a world model afterwards. This yields a position at the last cycle and a velocity vector. By multiplying the velocity vector by the cycle time and adding it to the position, the current position in world coordinates can be determined. This position gets reprojected into the image, yielding another candidate.

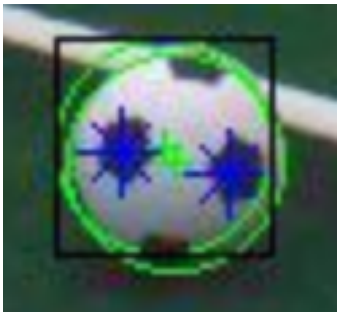

Figure 3.4: *Reprojected ball candidate (green circle within black bounding box). After reprojecting the tracked measurements of the ball filter, a new candidate for the classifier is generated.*

3.1.5 Saving the candidates

The generated candidates can be directly written to the disc on the NAO, with the estimated class in the filename for evaluation purposes. While there is no classifier in the beginning of the project, all candidates will be labeled as ‘false’.

3.2 Labeling

After collecting the candidates during test games, they have to be labeled for training. For this purpose a tool called ‘tinball’ was implemented to delegate the labeling process to the whole hulks team.

The tool shows a grid of nine images and preselects them according to the result of the classifier used during the generation. At first, images which are not yet labeled get randomly selected. If there not enough of those images for display, already labeled images get selected randomly. It is possible that one image is labeled multiple times, and every label is logged. This enables the possibility to find edge cases more easily.

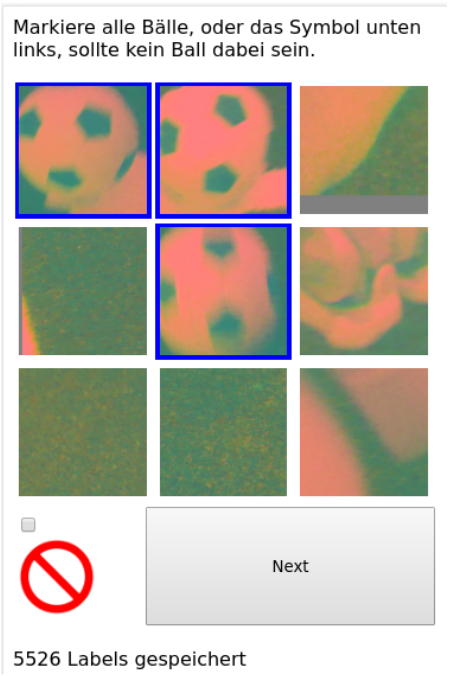

Figure 3.5: *The labeling tool ’tinball’. Images are saved in YCbCr and interpreted as RGB. The classes determined by the classifier during generation are pre-labeled.*

3.3 Data Setup

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

- RoboCup 2016 (Leipzig)
- Iran Open 2017 (Teheran)
- German Open 2017 (Magdeburg)
- Weekly Test Games (HULKs Laboratories)

The data used during the development of the training consisted of 16880 positive examples (candidate images containing a ball) and 23876 negative examples (candidate images not containing a ball). During the training, the negative examples are subsampled randomly, such that in every cross-validation set the same amout of positive and negative examples are present.

The final network derived in this thesis was evaluated with data collected at a testing event in the laboraties of HULKs mixed-team partner B-Human.

This can be considered as a proper generalization test because no data from these testing conditions was used during the development. The test data contained 5687 postive and 12730 negative examples.

Chapter 4

Genetic design of CNNs

The topology of the networks was optimised using the genetic algorithm mentioned in section genetic algorithm. Therefore, a network topology refers to an indiviual within the search space.

4.1 CNN Structure

The following common structure is used for creating all networks in the search space. First, the input image is resized to a fixed size using nearest neighbour interpolation. Then, multiple convolutional layers, each with an defined mask size are applied. Between every convolutional layer, a pooling layer with an activation function may be applied. The next step is a batch normalization layer. Finally, a fully connected network with a certain depth of hidden layers is applied. The general structure of a network is illustrated in figure 4.1.

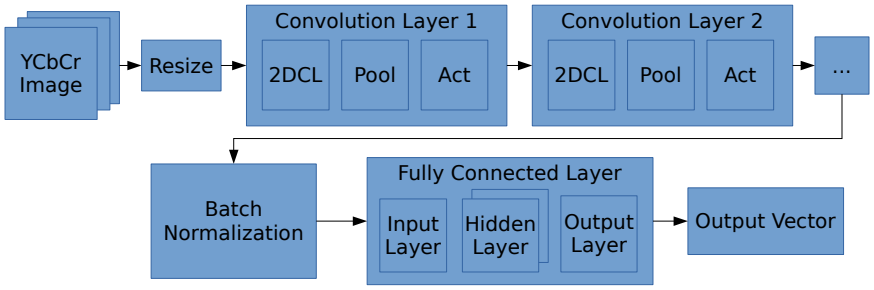

Figure 4.1: *The general structure of a CNN used in the experiments. First, the image gets resized to a a quadratic size. Then, for each convolutional layer a two dimensional convolution mask is applied (2DCL) followed by a pooling layer (Pool) and an activation function (Act). After normalizing the resulting image, a multi-layer ANN is applied, yielding the final output vector.*

There are many degrees of freedom (dof) for this topology of a network. The resizing layer has one dof, as the image gets resized to a quadratic shape. For each convolutional layer, there are five degrees of freedom: the number of output channels, the mask size in both dimensions, as well as the used pooling type and activation function. Finally, the dof of the fully connected ANN consist of the amount and size of the hidden layers, and the used activation function.

4.2 Search space

The search space for the genetic algorithm consisted of the parameters described in section CNN structure with the value range described in the following. The input image was sampled within the range $[8, 16] \in \mathbb{N}$. The amount of convolutional layers was limited to two. For each layer, there have been five layers at maximum with sizes within $[2, 3] \in \mathbb{N}$ 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 ANN, each with a number of neurons within $[2, 20] \in \mathbb{N}$.

4.3 Fitness function

The fitness function 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

For each network, a k-fold cross validation will be executed, yielding k values of tnr_{nk}, tpr_{nk} of that network n. In order to approximate a lower bound of these perfomance metrics, the difference of the mean and the variance is used for the fitness function. The tpr_n for a network n is computed by:

$$tpr_n = \text{mean}(tpr_{n1}, ..., tpr_{nk}) - \text{var}(tpr_{n1}, ..., tpr_{nk}) \tag{4.1}$$

, 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:

$$tnr_n = \text{mean}(tnr_{n1}, ..., tnr_{nk}) - \text{var}(tnr_{n1}, ..., tnr_{nk}) \tag{4.2}$$

4.3.2 Inference complexity

The complexity of a network is asymptotically approximated and lineary scaled. The complexity cc of a convolutional layer i is approximated by

$$cc_i = \frac{I_x \cdot I_y \cdot I_c \cdot m_x \cdot m_y \cdot m_c}{\hat{I}_x \cdot \hat{I}_y \cdot I_c \cdot \hat{m}_x \cdot \hat{m}_y \cdot \hat{m}_c} = \frac{I_x \cdot I_y \cdot m_x \cdot m_y \cdot m_c}{\hat{I}_x \cdot \hat{I}_y \cdot \hat{m}_x \cdot \hat{m}_y \cdot \hat{m}_c}$$

{#eq:comp-c},

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, while the $\hat{I}_x, \hat{I}_y, \hat{m}_x, \hat{m}_y, \hat{m}_c$ correspond to the maximum values as defined in the search space.

The complexity cf of the fully connected part is approximated by

$$cf = \frac{\sum_{i=1}^{k} s_i \cdot s_{i-1}}{\sum_{i=1}^{k} \hat{s}_i \cdot \hat{s}_{i-1}} \tag{4.3}$$

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

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

$$c_n = 1 - \frac{\sum_{i=1}^{k} cc_i + cf}{k+1} \tag{4.4}$$

4.3.3 Resulting fitness function

Given the approximation of the classification performance and the inference complexity, the resulting fitness function is chosen as follows.

$$f_n = 0.7 \cdot tnr_n^2 + 0.25 \cdot tpr_n^2 + 0.05 \cdot c_n \tag{4.5}$$

As described in the goals section, 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.

4.4 Inference on the NAO

The Tensorflow framework does not support x86 architectures like the one used on the NAO. Therefore it is not possible to inference trained models directly on the robot. The needed functions for inferencing a network where implemented within the HULKs framework as a part of this thesis and will be with included in the HULKs code release of 2017.

The saved candidates where resized using nearest neighbour interpolation. On the robot, the network input is directly sampled from the full image with the same interpolation mode. Pixels which are outside of the image will be defaulted to zero input.

As defined in section normalization layer, the parameters for the batch normalization where approximated by a moving average approach during the training. Given these parameters, the normalization output can be calculated.

The convolutional layers were implemented straight forward according to the definition in tensorflow API documentation [4]. The output values are calculated according to {#eq:conv2d}. The pooling functions for average and maximum pooling are implemented for fixed pooling sizes, namely 2x2. During training, the default strategy ‘SAME’ was chosen for padding the image borders. In the Tensorflow framework, this

means basically zero padding, meaning the value zero is chosen for pixels outside of the image.

For the matrix multiplication of the fully connected layers, simply two nested for loops are used. For the hyperbolic tangend activation function, the implementation of the C++ standard library is used. The relu activation is implemented as defined with a simple if statement.

The code of the implementation can be found on the CD in the directory `Projects/nao`.

Chapter 5

Experiments and Evaluation

This chapter describes experiments and results of the genetic algorithm. In every experiment, 15 generations with 50 networks in each generation were evaluated. The worst 10% in each generation were excluded from reproduction. The mutation probality were chosen as $\frac{1}{16}$ according to the maximum number of degrees of freedom with the given search space.

The algorithm should be able to find a proper network as a solution to the ball detection problem, given a certain amount of training data. To show that this can be achieved, there were 4 Experiments, with step wise increased amount of data, randomly sampled from the whole training data. The experiments were executed with 25%, 50%, 75% and 100% of the available training data.

5.1 Experiment 1

In the first experiment, only 25% of the available data was used. The best networks in the last generation reached a tnr of 91% to 95% with a tpr from about 50% to 75%. The resulting network has a very small sample size of 8x8, one convolutional layer of 4 masks and only 3 hidden layers in the fully connected part. These networks where highly biased to reject the input, yielding a high tnr, while having a poor tpr. Figure 5.1 shows the densities of the components of the fitness function as well as the result of the fitness function itself.

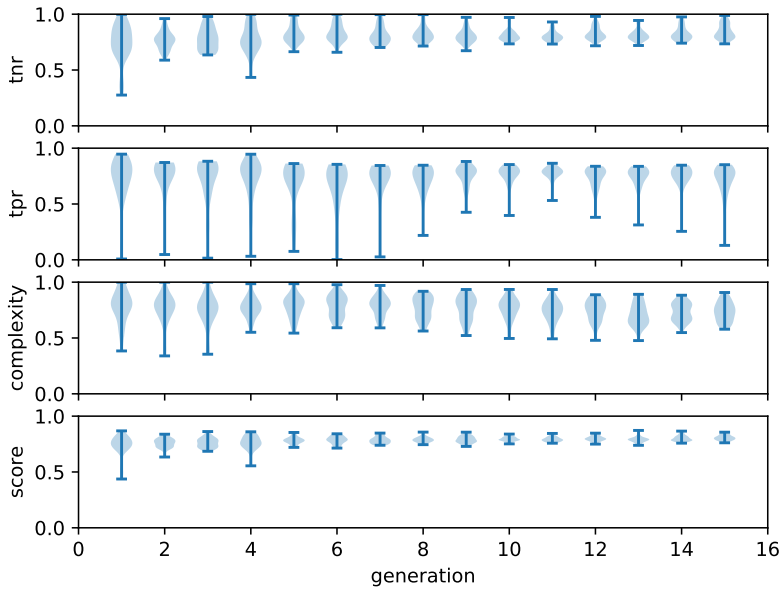

Figure 5.1: *Result densities of the fitness function over generations. The tnr plot shows the densities of the true negative rates lower bound (eq. 4.2) of each generation. The tpr plot shows the densities of the true positive rates lower bound (eq. 4.1) of each generation. The complexity plot shows the densities of the complexity penalty (eq. 4.4) of each generation. The score plot shows the densities of the fitness function results.*

While the variance of the tnr got smaller over time, the tpr seemed to got worse. The networks with a very high tnr in the first generation had also a very low tpr, as figure 5.2 illustrates. These individuals were therefore eliminated due to the weights of the fitness function.

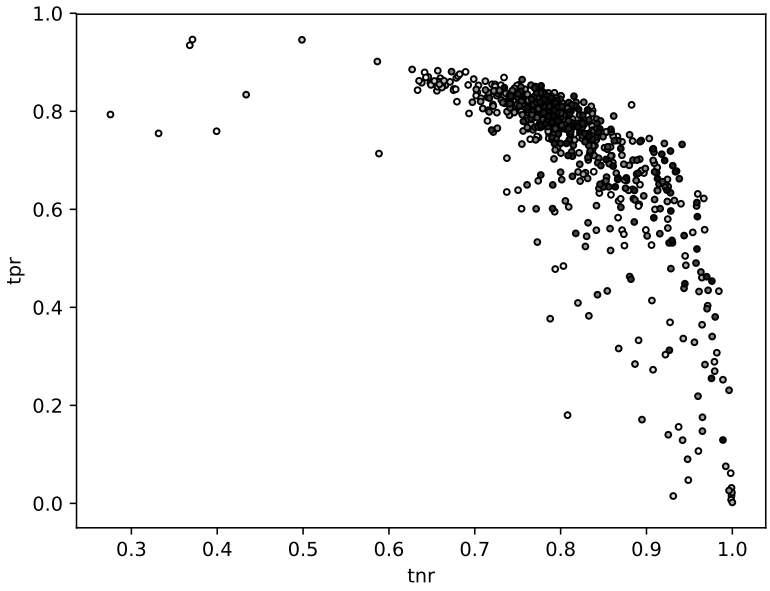

Figure 5.2: *Classification performance of the networks over generations. Results of the first generation are plotted with white filled circles. Results of the following generations are plotted in ascending darker colours.*

5.1.1 Population settings

One can see in table A.1, the sample size was nearly equally distributed in the end.

The second convolutional layers extinct very quickly, as shown in the tables A.2 and A.3. From generation 3 on, there where only a few individuals with two layers left. In generation 5, networks with two convolutional layers died out completely.

Table A.4 shows, also the hidden layer in the fully connected layer extinced very quickly. The last individuals having hidden layer existed in generation 12. From this generation on, the networks resulted in a single linear equtation from the full sampled image to the output vector with relu activation afterwards.

5.2 Experiment 2

In this experiment, 50% of the available training data was used. The best networks in the last generation reached a tnr of about 93% with a tpr above 85%. Networks with

one large convolutional layer and mainly 3 hidden layer in the fully connected parted dominated this experiment.

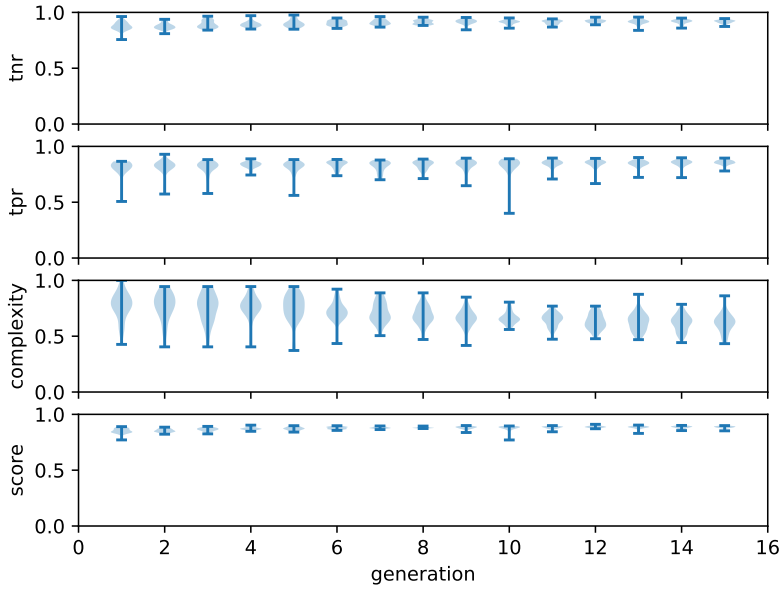

Figure 5.3: *Result densities of the fitness function over generations. The tnr plot shows the densities of the true negative rates lower bound (eq. 4.2) of each generation. The tpr plot shows the densities of the true positive rates lower bound (eq. 4.1) of each generation. The complexity plot shows the densities of the complexity penalty (eq. 4.4) of each generation. The score plot shows the densities of the fitness function results.*

Figure 5.3 shows that the scores slightly increased up to generation 4. While the complexity slightly increased over time, the densities of the tnr and tpr kept steady, only outliers in the tpr got eliminated.

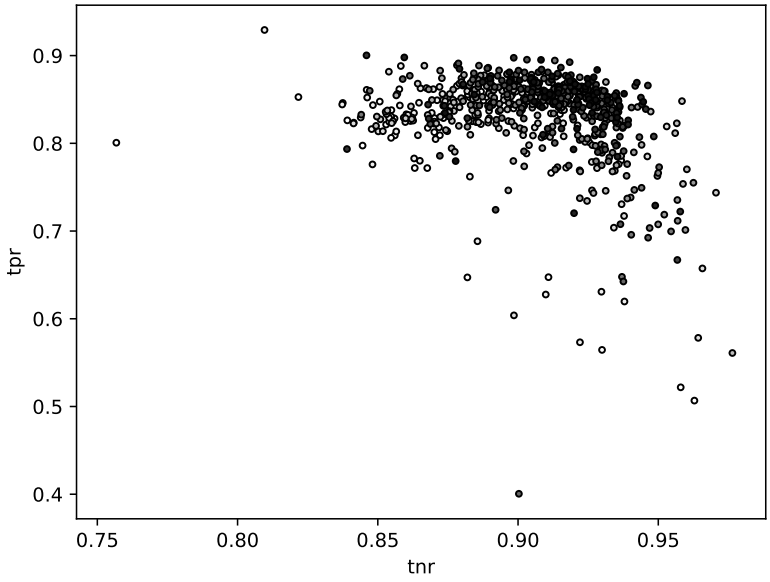

Figure 5.4: *Classification performance of the networks over generations. Results of the first generation are plotted with white filled circles. Results of the following generations are plotted in ascending darker colours.*

The classification performance was already at a very higher scale than in the first experiment, as illustrated by figure 5.3 and 5.1. Therefore, not only the individuals with a bad tnr were eliminated, but also the tpr converged over time. Figure 5.4 shows that there were no outliers with a tpr below 70% in the later generations.

5.2.1 Population settings

One can see in table A.5, the sample size seemed to converge against 12 in the end.

Tables A.6 and A.7 show that the number of convolutional layers clearly converged against 1. From generation 9 on, only those networks survived. Within this one layer, the majority of the resulting networks had 5 masks. Most of those masks were of size 3x3. For the activation and pooling function, clearly relu and max pooling dominated in this experiment.

As already in the convolutional layer, relu activation clearly dominated in the fully connected part, as one can see in table A.8. Most of the networks had 3 hidden layer in the end.

5.3 Experiment 3

In this experiment, 75% of the available training data was used. The best networks in the last generation reached a tnr of about 94% with a tpr above 88%. The best rated networks in the last generation were slightly more complex, having a higher sample size and more hidden layers than in experiment 2.

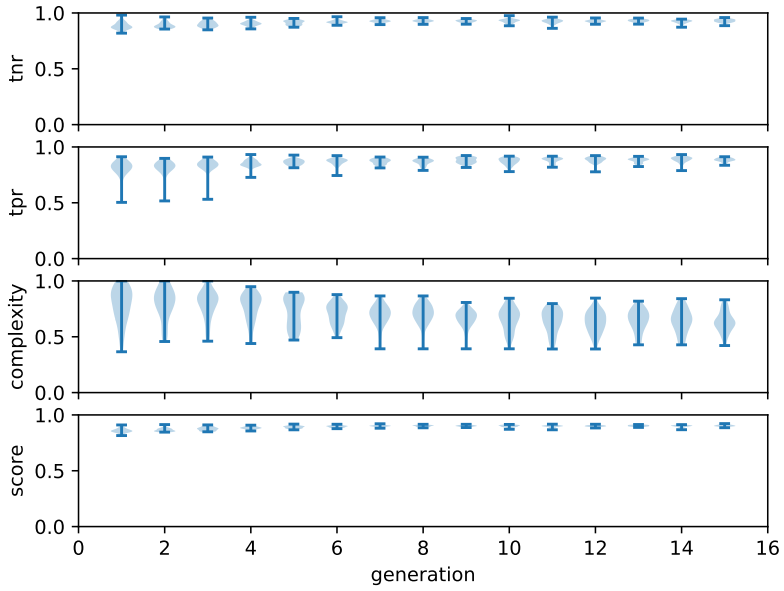

Figure 5.5: *Result densities of the fitness function over generations. The tnr plot shows the densities of the true negative rates lower bound (eq. 4.2) of each generation. The tpr plot shows the densities of the true positive rates lower bound (eq. 4.1) of each generation. The complexity plot shows the densities of the complexity penalty (eq. 4.4) of each generation. The score plot shows the densities of the fitness function results.*

Figure 5.5 shows that the scores slightly increased up to generation 4. The complexity slightly increased over time. The densities of tnr and tpr kept steady, only outliers in the tpr got eliminated.

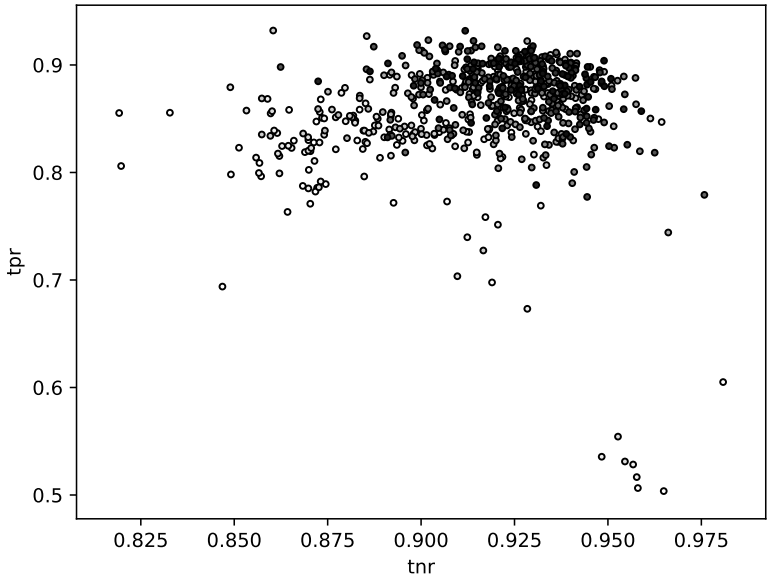

Figure 5.6: *Classification performance of the networks over generations. Results of the first generation are plotted with white filled circles. Results of the following generations are plotted in ascending darker colours.*

The classification performance converged very quickly in this experiment. The later generations built a very dense cluster when looking at the correlation plot of figure 5.6.

5.3.1 Population settings

As table A.9 shows, the sample size seemed to converge against 11 in the end.

Tables A.10 and A.11 show that the number of convolutional layers clearly converged against 1. Within this one layer, the majority of the resulting networks had 5 masks. From generation 9 on, only those networks survived. Most of those masks were of size 3x3. For the activation and pooling function, it clearly relu and max pooling dominated in this experiment.

As already in the convolutional layer, relu activation clearly dominated in the fully connected part, as table A.12 shows. Most of the networks had 3 hidden layer in the end.

5.4 Experiment 4

In this experiment, all of the available training data was used. The best networks in the last generation reached a tnr of about 95% with a tpr above 90%. While the sample size and convolutional layers are quite the same as in Experiment 3, the hidden layer part converged against 4 hidden layers instead of 3.

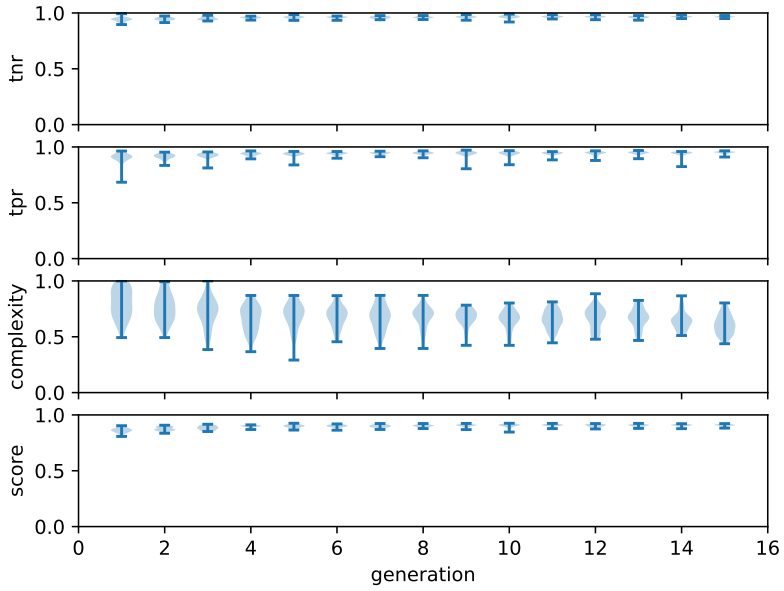

Figure 5.7: *Result densities of the fitness function over generations. The tnr plot shows the densities of the true negative rates lower bound (eq. 4.2) of each generation. The tpr plot shows the densities of the true positive rates lower bound (eq. 4.1) of each generation. The complexity plot shows the densities of the complexity penalty (eq. 4.4) of each generation. The score plot shows the densities of the fitness function results.*

As Figure 5.7 shows, the distribution of the scores did not increase after the 6th generation. While the distributions of the tpr and tpr seemed very constant, figure 5.8 shows that the later generations built again a very dense cluster in the correlation plot. Hence the algorithm could not really find a much better solution throughout the generations, but was able to select the best ones and remove the outliers.

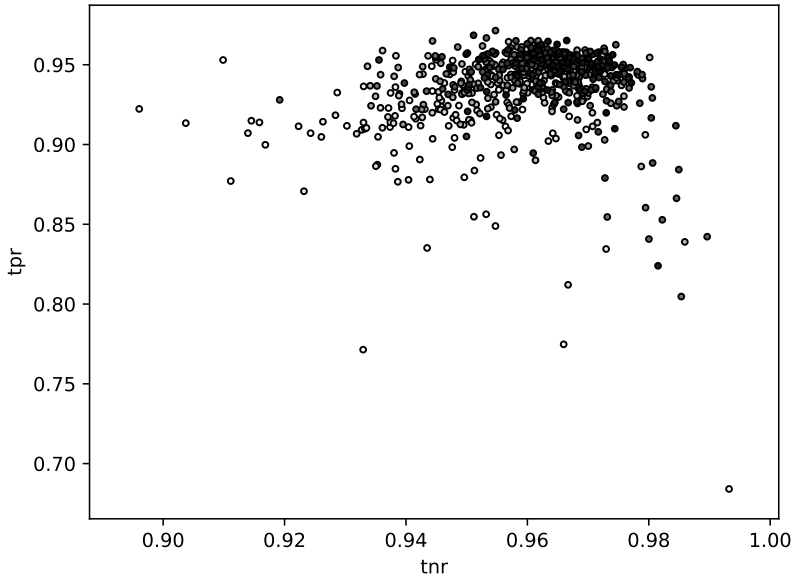

Figure 5.8: *Classification performance of the networks over generations. Results of the first generation are plotted with white filled circles. Results of the following generations are plotted in ascending darker colours.*

5.4.1 Population settings

Looking at the histograms of table A.13, the sample size of the generated candidates has a bimodal charactor from generation 8 until the last. The peaks in the last generation where at sample size 11 and 15.

Tables A.14 and A.15 show that the number of convolutional layers clearly converged against 1. Within this one layer, the majority of the resulting networks had 5 masks. From generation 8 on, only those networks survived. Considering the size of those masks, it seems that the neighborhood of image pixels in x is more important than in y direction. The classification performance is indifferent between maximum pooling or no pooling. As the maximum pooling reduces the inference complexity due to the smaller input size into the fully connected layer, the networks with max pooling where preferred in the end.

As already in the convolutional layer, also in the fully connected part the relu activation function clearly dominates, as table A.16 illustrates. From generation 8 on, every

network contained 4 hidden layer. Still, the fully connected parts of the dominating networks reduce the complexity within the network and enlarge it again.

5.5 Evaluation

The best network of the last generation is considered as the resulting network of the algorithm. Table 5.1 shows a summary of the resulting networks in each experiment. The resulting networks became more complex with increasing amount of data, yielding better classification performance.

Table 5.1: *Overview of the resulting networks of the Experiments. The column 'exp' lists the number of the experiment. The second column 'used data' corresponds to the amount of data used in this experiment. The columns 'tnr', 'tpr' and 'comp' show the components of the fitness function. The column 'score' corresponds to the resulting score.*

exp	used data	tnr	tpr	comp	score
1	25%	0.921	0.7	0.732	0.856
2	50%	0.932	0.853	0.663	0.899
3	75%	0.949	0.904	0.64	0.922
4	100%	0.972	0.958	0.638	0.922

In summary, the experiments confirm the expected behavior of finding a proper network as a solution to the ball detection problem, given a certain amount of training data. They also show the advantage that no manual design of the network is needed. Still, the optimization of the architecture needs a lot of computation time, as the algorithm has to train and evaluate (number of cross-validation subsets) $\cdot$ (number of individuals per generation) $\cdot$ (number of generations) CNNs, in our case 2250 networks.

Chapter 6

Further Evaluation

After running the experiments, the topology of the resulting network of the last experiment was further evaluated. A network with this topology was trained with the complete dataset. This network was used to investigate it's classification performance even further by optimizing the input and a generalization test.

6.1 Input Optimization

The tensorflow graph of the network has been modified, with the weight matrices and output as constants and the input image as trainable variable. Figure 6.1 shows the optimization with a real candidate as initialization of the input variable. Figure 6.2 shows the same optimization, but with a zero initialized input.

In figure 6.1, there is slightly more heating in the lower part of the ball. In figure 6.2, the image got cooled in the middle part and a view surrounding pixels got heated up, which could correspond to a dark patch.

In summary, the input optimization does not yield really much information about what is happening within the classifier. That result is not very supprising, as such 'dream' images are rarely usefull on such small networks.

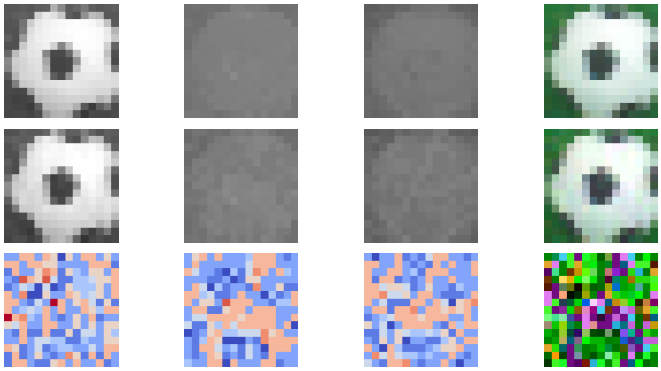

Figure 6.1: *The result of the input optimization given an image of ball as initial input. The upper line shows plots of the y, cb, cr channel and the corresponding rgb image of the initial input. The middle line shows the optimized input image in the same manner. The last row shows heat plots of the difference in the y, cb, cr channel, as well as an rgb interpretation of the difference.*

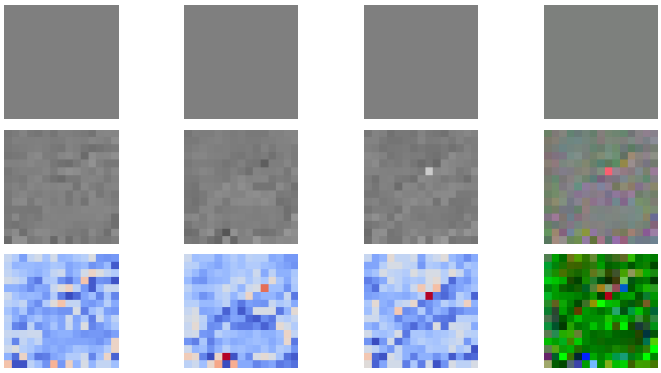

Figure 6.2: *The result of the input optimization given a zero initialized input. The upper line shows plots of the y, cb, cr channel and the corresponding rgb image of the initial input. The middle line shows the optimized input image in the same manner. The last row shows heat plots of the difference in the y, cb, cr channel, as well as an rgb interpretation of the difference.*

6.2 Generalization Test

A few weeks before the RoboCup 2017, the HULKs Team attended at a workshop at the University of Bremen together with B-Human. As this was a complete new environment, testing the classifier there would be a proper generalization test.

During a test game against B-Human, 18417 images were collected, together with the predicted class by the network. After labeling those images, the resulting confusion matrix is:

	positives	negatives
positive predict	4989	50
negative predict	698	12680

Resulting in a true positive rate of 87,73% and a true negative rate of 99,61%.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

The aim of this work was to find an automated way for a CNN based ball detection on the NAO robotic system. While implementing a genetic algorithm for designing such a CNN it was possible to find a network which yields proper classification. By implementing the nessecary inference functions for the model, it was possible to use the classifier on the NAO during the RoboCup 2017.

From the results in the Experiments, one can see that the genetic algorithm designs very small, still good working networks according to the amount of data which is given. However, the candidate generation could not be evaluated due to the missing ground truth data. The calculated results of the classifer only express the performance given the generated examples.

The most important part of solving images processing problems with CNNs is still the setup of a proper candidate generation and data set, as well as the evaluation of the training results. Still, the algorithm lead to good results which could be used in the RoboCup Competitions 2017.

7.2 Outlook

As mentioned in the Conclusion, the candidate generation could not be evaluated. For future work on machine learning approaches, a better tooling for labeling whole images would be nessesary.

After the latest success of the HULKs team at the RoboCup 2017, a working robot detection is more urgent. As the inference of CNNs is very expensive, it will probably

not possible to just apply the network from Chris Kahlefendts project thesis [14] in addition to this ball detector. The genetic algorithm could be extended to be able to design multi class problems and the solve the ball and the robot detection within one network.

Given a multiclass detector, the algorithm can be evaluated on problems outside the RoboCup Context, for example on MNIST [17] or CIFAR-10 [16] to be comparable to state of the art solutions.

References

[1] Rodrigo Benenson. *Classification datasets results*. Feb. 2016. URL: https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

[2] B-Human. "Team Description for RoboCup 2016". In: (2016). URL: https://www.b-human.de/downloads/publications/2016/TDP2016.pdf.

[3] Google Developers. *About TensorFlow*. Aug. 2017. URL: https://www.tensorflow.org/#about-tensorflow.

[4] Google Developers. *Tensorflow API Doc, tf.nn.conv2d*. Aug. 2017. URL: https://www.tensorflow.org/api_docs/python/tf/nn/conv2d.

[5] SoftBank Robotics Europe. *NAO Humanoid Robot Platform Datasheet*. 2017. URL: https://www.ald.softbankrobotics.com/sites/aldebaran/files/datasheet_nao_next_gen_en.pdf.

[6] SoftBank Robotics Europe. *Who is NAO?* 2017. URL: https://www.ald.softbankrobotics.com/en/robots/nao.

[7] RoboCup Federation. *A Brief History of RoboCup*. 2016. URL: http://robocup.org/a_brief_history_of_robocup.

[8] RoboCup Federation. *RoboCupSoccer - Standard Platform*. 2016. URL: http://robocup.org/leagues/5.

[9] Fabian Flöck. Sept. 2016. URL: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#/media/File:K-fold_cross_validation_EN.jpg.

[10] Steffen Harbich. "Einführung genetischer Algorithmen mit Anwendungsbeispiel". In: (Dec. 2007).

[11] Timm Hess et al. "Large-scale Stochastic Scene Generation and Semantic Annotation for Deep Convolutional Neural Network Training in the RoboCup SPL". In: (July 2017). URL: https://pdfwww.gakkai-web.net/gakkai/inter/robocup2017symposium/contents/html/papers/pdf/RoboCup_Symposium_2017_paper_15.pdf.

[12] Nao-Team HTWK. “Team Research Report 2016”. In: (2016). URL: http://robocup.imn.htwk-leipzig.de/documents/TRR_2016.pdf.

[13] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: (Mar. 2015).

[14] Chris Kahlefendt. “A Comparison and Evaluation of Neural Network-based Classification Approaches for the Purpose of a Robot Detection on the Nao Robotic System”. Apr. 2017. URL: http://www.hulks.de/_files/PA_Chris-Kahlefendt.pdf.

[15] Ron Kohavi. “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection”. In: (1995).

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *The CIFAR-10 dataset*. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[17] Yann LeCun, Corinna Cortes, and Cristopher J.C. Burges. *THE MNIST DATABASE of handwritten digits*. 2017. URL: http://yann.lecun.com/exdb/mnist/.

[18] Nathapon Olaf Lüders. “Object Localization on the Nao Robotic System Using a Deep Convolutional Neural Network and an Image Contrast Based Approach”. Aug. 2016. URL: https://www.hulks.de/_files/MA_Olaf_Lueders.pdf.

[19] Melanie Mitchel. *An Introduction to Genetic Algorithms*. A Bradford Book, The MIT Press, 1999. ISBN: 0-262-63185-7.

[20] W.K. Purves et al. *Biologie*. Elsevier Spektrum Akademiker Verlag, 2004. ISBN: 978-3-8274-2007-7.

Appendix A

Single results of the Networks in the Experiments

A.1 Detailed results of experiment 1

Table A.1: *Histograms over the sample size in every generation. Column '#' shows the number of the generation. Column 'sample size' lists the histograms. The left-most value corresponds to the number of networks having the lowest sample size. The right-most value to the networks having the highest sample size.*

#	sample size
0	(4,4,6,4,6,3,6,10,7)
1	(2,1,5,3,6,7,5,18,3)
2	(3,1,3,4,7,11,4,14,3)
3	(2,0,1,7,7,3,8,17,5)
4	(2,0,0,9,8,0,9,16,6)
5	(3,0,0,10,4,0,10,16,7)
6	(2,0,0,11,2,0,9,18,8)
7	(1,2,2,18,0,1,8,12,6)
8	(2,0,2,10,2,4,12,8,10)
9	(6,0,3,12,2,4,6,7,10)
10	(6,0,4,7,2,9,9,5,8)
11	(10,2,4,10,2,5,7,2,8)
12	(6,1,7,10,1,9,6,0,10)
13	(8,0,6,10,1,6,10,1,8)
14	(8,3,8,5,4,7,8,0,7)

Table A.2: *Histograms over the settings for convolutional layer 1 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(17,18)	(11,12,12)	(14,21)	(17,18)	(8,9,7,9,2)
1	(19,20)	(13,7,19)	(19,20)	(16,23)	(11,12,7,8,1)
2	(21,18)	(14,5,20)	(21,18)	(15,24)	(12,11,7,8,1)
3	(15,25)	(7,7,26)	(20,20)	(20,20)	(15,8,7,7,3)
4	(25,20)	(7,5,33)	(27,18)	(23,22)	(22,7,7,8,1)
5	(24,24)	(8,3,37)	(24,24)	(21,27)	(18,14,8,6,2)
6	(27,23)	(10,4,36)	(25,25)	(27,23)	(18,16,6,8,2)
7	(28,22)	(18,1,31)	(25,25)	(32,18)	(13,15,11,9,2)
8	(23,27)	(25,2,23)	(19,31)	(37,13)	(10,25,4,8,3)
9	(22,28)	(25,4,21)	(14,36)	(33,17)	(9,24,4,6,7)
10	(23,27)	(31,2,17)	(17,33)	(32,18)	(3,21,3,12,11)
11	(20,30)	(32,0,18)	(20,30)	(31,19)	(3,27,2,9,9)
12	(21,29)	(35,1,14)	(16,34)	(34,16)	(0,23,5,12,10)
13	(15,35)	(35,2,13)	(18,32)	(33,17)	(0,24,4,12,10)
14	(11,39)	(37,3,10)	(15,35)	(24,26)	(2,26,5,11,6)

Table A.3: *Histograms over the settings for convolutional layer 2 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(10,4)	(4,3,7)	(5,9)	(4,10)	(2,2,3,1,6)
1	(9,2)	(4,2,5)	(3,8)	(7,4)	(0,6,2,0,3)
2	(5,6)	(3,4,4)	(3,8)	(4,7)	(0,3,2,0,6)
3	(2,1)	(1,1,1)	(0,3)	(2,1)	(0,1,0,0,2)
4	(2,0)	(0,0,2)	(0,2)	(2,0)	(0,2,0,0,0)

#	activation	pooling	size_x	size_y	masks
5	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
6	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
7	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
8	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
9	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
10	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
11	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
12	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
13	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
14	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)

Table A.4: *Histograms over the settings for the fully connected layer in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The layer column lists the histograms of the size in every possible layer.*

#	activation	layer
0	(24,26)	(2,2,1,1,2,2,2,2,2,4,4,2,3,4,4,1,0,3,0) (1,3,4,2,1,2,1,3,4,1,1,0,1,1,1,1,1,4,2) (1,0,0,2,2,3,0,1,1,1,0,1,2,2,3,0,1,0,4) (0,1,2,2,0,1,1,0,1,1,2,3,0,0,0,0,1,1,1)
1	(30,20)	(0,1,0,0,0,6,3,0,1,6,5,2,2,8,7,1,1,2,0) (0,3,2,1,0,0,2,4,6,1,2,0,2,4,0,1,1,6,1) (1,0,0,2,2,3,0,0,1,1,0,0,4,1,5,1,0,0,4) (0,0,0,4,0,4,2,0,1,4,1,0,0,0,0,1,2,1,1)
2	(28,22)	(0,0,1,0,0,6,2,0,1,4,4,3,1,9,8,1,0,7,0) (0,4,1,0,0,0,3,2,11,0,3,0,1,6,0,1,0,7,0) (0,0,0,2,3,3,0,0,0,5,0,0,5,0,5,2,0,0,4) (0,0,0,6,0,6,3,0,1,5,3,0,0,0,0,0,1,1,0)
3	(22,28)	(0,0,2,0,0,7,1,0,2,4,5,2,0,9,4,2,0,9,0) (0,1,1,0,0,0,4,2,10,0,1,0,1,7,1,1,0,8,0) (0,1,0,3,4,2,0,0,0,4,0,0,2,0,4,3,0,0,4) (0,0,0,5,1,5,4,0,1,4,2,0,0,0,0,0,0,0,0)
4	(24,26)	(1,1,1,1,0,5,2,0,0,3,6,3,0,4,4,3,0,12,1) (0,1,0,0,0,0,4,0,7,0,3,1,2,5,0,0,0,9,0) (0,0,0,1,4,2,1,0,0,3,0,0,3,0,1,4,0,0,3) (0,0,0,3,0,6,4,0,0,1,3,0,0,0,0,0,0,0,0)
5	(23,27)	(0,1,0,2,1,6,5,0,0,2,5,0,1,3,4,3,1,12,0)

#	activation	layer
		(1,3,0,0,0,1,0,0,3,0,2,0,2,4,1,0,0,10,0)
		(0,0,0,1,0,2,0,0,0,2,0,0,6,0,0,3,0,0,2)
		(0,0,0,2,0,3,0,0,0,1,6,0,0,0,0,0,0,0,0)
6	(32,18)	(0,0,0,3,0,3,10,0,0,0,4,0,0,5,8,5,0,9,0)
		(2,2,1,0,0,4,0,0,4,0,0,0,5,3,0,0,0,8,0)
		(0,0,0,2,0,6,0,0,0,3,0,0,5,0,0,1,0,0,4)
		(0,0,0,3,0,1,0,0,0,2,5,0,0,0,0,0,0,0,0)
7	(30,20)	(0,0,0,1,0,1,13,1,0,0,0,0,0,6,8,4,0,14,0)
		(0,0,0,0,0,1,0,0,5,0,0,0,7,2,0,0,0,9,0)
		(1,0,0,1,0,7,0,0,0,5,0,0,0,0,0,1,0,0,0)
		(0,0,0,4,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0)
8	(31,19)	(0,0,0,0,0,1,23,1,0,0,0,1,0,3,9,4,0,7,0)
		(0,0,0,0,0,1,0,0,2,0,0,0,8,0,1,0,0,8,0)
		(0,0,0,0,0,8,0,0,0,3,0,0,0,0,0,1,0,0,0)
		(0,0,0,1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0)
9	(34,16)	(1,0,0,0,0,0,25,0,0,0,0,0,0,3,9,3,0,9,0)
		(0,0,0,1,0,4,0,0,3,0,0,0,5,0,2,0,0,5,0)
		(0,0,0,0,0,9,0,0,0,5,0,0,0,0,0,0,0,0,0)
		(0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0)
10	(36,14)	(0,0,0,1,0,1,22,0,0,0,1,0,0,3,9,5,0,8,0)
		(0,0,0,1,0,9,0,0,0,0,0,0,1,0,3,0,0,7,1)
		(1,0,1,0,1,8,0,0,0,2,0,0,0,0,0,0,0,0,0)
		(0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0)
11	(30,20)	(0,1,0,0,0,0,18,0,0,1,0,0,0,1,12,8,0,9,0)
		(0,0,0,0,0,11,0,0,0,0,0,0,1,0,4,0,0,8,0)
		(1,0,2,0,0,9,0,0,1,3,0,0,0,0,0,0,0,0,0)
		(0,0,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0)
12	(27,23)	(0,0,0,1,1,0,11,0,0,4,0,0,0,3,11,3,1,15,0)
		(0,0,0,0,1,8,0,0,0,0,0,0,0,1,3,0,0,6,0)
		(0,0,3,0,0,6,0,0,2,1,0,0,0,0,0,0,1,0,0)
		(0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0)
13	(25,25)	(0,0,0,2,0,0,13,0,0,3,0,0,0,3,15,2,1,11,0)
		(0,0,0,0,3,7,0,0,0,0,0,0,0,0,5,0,0,4,1)
		(0,0,3,0,0,5,0,0,3,1,0,1,0,0,0,0,3,0,0)
		(0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0)
14	(21,29)	(0,0,0,3,0,0,10,0,0,2,0,0,1,1,18,2,0,13,0)
		(0,0,0,0,6,8,0,0,0,0,0,0,0,0,7,0,0,2,0)
		(0,0,4,0,0,7,0,0,4,1,0,1,0,0,0,0,0,4,0,0)

#	activation	layer
		(0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0)

A.2 Detailed results of experiment 2

Table A.5: *Histograms over the sample size in every generation. Column '#' shows the number of the generation. Column 'sample size' lists the histograms. The left-most value corresponds to the number of networks having the lowest sample size. The right-most value to the networks having the highest sample size.*

#	sample size
0	(6,4,6,3,6,7,7,6,5)
1	(5,6,5,6,8,1,8,5,6)
2	(2,6,6,3,3,1,13,7,9)
3	(2,2,5,3,1,0,16,9,12)
4	(2,4,8,0,3,0,15,9,9)
5	(2,2,2,0,1,0,15,14,14)
6	(2,3,2,1,0,0,19,13,10)
7	(0,7,0,4,1,0,15,16,7)
8	(1,6,0,3,3,1,13,19,4)
9	(1,1,0,4,6,3,10,21,4)
10	(1,2,0,4,9,4,6,22,2)
11	(0,3,0,4,15,3,7,16,2)
12	(0,6,0,3,16,2,10,13,0)
13	(0,5,1,6,15,3,14,6,0)
14	(0,4,0,7,19,1,13,4,2)

Table A.6: *Histograms over the settings for convolutional layer 1 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(17,18)	(15,6,14)	(14,21)	(17,18)	(7,7,4,11,6)

#	activation	pooling	size_x	size_y	masks
1	(17,16)	(13,9,11)	(14,19)	(14,19)	(5,6,2,15,5)
2	(17,14)	(12,13,6)	(13,18)	(19,12)	(3,3,1,24,0)
3	(12,15)	(7,14,6)	(14,13)	(19,8)	(0,3,2,22,0)
4	(15,17)	(11,17,4)	(17,15)	(17,15)	(0,0,5,27,0)
5	(13,22)	(11,16,8)	(22,13)	(22,13)	(0,0,5,30,0)
6	(18,18)	(15,13,8)	(18,18)	(21,15)	(0,0,8,28,0)
7	(18,21)	(14,17,8)	(13,26)	(26,13)	(0,0,7,32,0)
8	(13,35)	(10,27,11)	(15,33)	(32,16)	(0,1,3,43,1)
9	(11,39)	(8,32,10)	(11,39)	(34,16)	(1,0,3,46,0)
10	(4,46)	(2,32,16)	(12,38)	(27,23)	(0,0,2,47,1)
11	(4,46)	(5,28,17)	(9,41)	(26,24)	(0,0,1,45,4)
12	(4,46)	(9,30,11)	(7,43)	(24,26)	(0,1,3,42,4)
13	(8,42)	(6,35,9)	(8,42)	(17,33)	(0,1,3,40,6)
14	(2,48)	(6,40,4)	(7,43)	(24,26)	(0,0,8,31,11)

Table A.7: *Histograms over the settings for convolutional layer 2 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(5,16)	(9,3,9)	(13,8)	(9,12)	(5,4,3,6,3)
1	(5,7)	(3,4,5)	(5,7)	(5,7)	(2,0,3,6,1)
2	(0,5)	(0,5,0)	(0,5)	(0,5)	(0,0,0,5,0)
3	(0,4)	(0,4,0)	(0,4)	(0,4)	(0,0,0,4,0)
4	(1,4)	(0,5,0)	(0,5)	(0,5)	(0,0,0,5,0)
5	(0,1)	(0,1,0)	(0,1)	(0,1)	(0,0,1,0,0)
6	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
7	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
8	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
9	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
10	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
11	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
12	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
13	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)

#	activation	pooling	size_x	size_y	masks
14	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)

Table A.8: *Histograms over the settings for the fully connected layer in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The layer column lists the histograms of the size in every possible layer.*

#	activation	layer
0	(18,32)	(2,1,4,3,2,3,4,4,3,2,1,1,3,2,0,3,0,3,2)
		(4,1,2,2,1,0,1,4,2,1,1,3,2,2,1,2,0,2,2)
		(1,2,1,4,1,2,2,2,0,1,2,0,1,2,2,0,0,2,1)
		(0,0,1,1,2,1,0,1,0,1,0,0,0,0,0,1,2,1,0)
1	(17,33)	(3,1,4,5,3,3,2,3,1,1,2,2,2,7,0,2,1,3,4)
		(5,1,5,3,2,0,0,6,0,1,1,5,3,4,0,1,0,2,2)
		(2,2,1,7,3,2,2,4,0,1,2,0,0,2,0,0,0,2,5)
		(0,0,1,1,1,3,0,1,0,1,0,0,0,1,0,0,1,0,0)
2	(11,39)	(1,0,7,3,0,4,1,6,2,1,7,1,1,8,0,1,2,1,4)
		(3,2,7,1,1,0,0,5,0,0,0,1,4,8,0,0,0,2,2)
		(0,2,1,8,1,4,1,4,0,0,0,0,0,7,0,0,0,3,3)
		(1,0,0,0,0,6,0,2,0,2,0,0,0,2,0,0,0,0,0)
3	(5,45)	(0,0,6,0,0,4,1,9,2,2,7,3,1,9,0,3,0,0,3)
		(0,3,7,0,0,0,0,5,0,0,0,0,3,10,1,0,0,2,0)
		(0,2,2,6,0,4,1,4,0,0,0,0,1,8,0,0,0,3,0)
		(0,0,0,0,0,10,0,1,0,2,0,0,0,0,0,0,0,0,0)
4	(3,47)	(0,0,6,1,0,7,0,5,3,2,9,1,0,10,0,1,0,1,4)
		(0,1,5,0,0,0,0,5,0,0,1,0,4,15,3,0,0,3,1)
		(0,3,0,7,0,5,0,8,0,0,0,0,1,10,0,0,1,3,0)
		(0,0,0,0,0,10,0,0,0,3,0,0,0,0,0,0,0,0,0)
5	(2,48)	(0,0,1,1,0,5,1,3,3,1,20,1,0,6,0,2,1,0,5)
		(0,2,2,0,1,0,1,4,0,0,1,0,5,23,1,0,0,6,0)
		(0,2,0,4,1,2,0,5,1,0,1,0,3,21,0,0,1,4,1)
		(0,0,0,0,0,22,0,0,0,2,0,0,1,0,0,0,0,0,0)
6	(1,49)	(0,0,0,1,0,2,4,2,3,1,20,1,0,7,0,1,0,0,8)
		(0,1,1,0,0,0,1,2,0,0,1,1,6,23,2,0,1,8,0)
		(0,2,0,6,1,1,1,3,0,0,1,0,1,22,0,0,0,8,1)
		(0,0,1,0,0,17,1,0,0,2,0,2,1,0,0,0,0,0,1)
7	(1,49)	(0,0,0,0,0,1,1,2,4,0,20,0,0,11,0,1,0,0,10)
		(0,0,0,1,0,0,0,0,0,1,2,0,7,23,3,0,1,12,0)

#	activation	layer
		(0,4,2,9,0,2,0,1,0,0,0,0,1,20,0,0,0,11,0)
		(0,0,2,0,0,15,0,0,0,1,0,3,2,1,0,0,1,0,0)
8	(2,48)	(0,0,0,0,0,0,1,1,1,0,29,0,0,10,0,0,0,0,8)
		(0,1,0,1,0,0,1,1,0,3,0,0,5,29,1,0,1,7,0)
		(1,1,1,8,0,1,0,0,0,0,0,0,0,29,0,0,0,9,0)
		(0,1,0,0,0,26,0,0,0,0,0,3,0,1,0,0,0,0,0)
9	(1,49)	(0,0,0,0,0,0,1,0,0,1,28,0,0,13,0,0,0,0,7)
		(0,1,0,1,0,0,0,2,0,7,1,0,1,27,0,0,1,9,0)
		(0,0,0,9,0,2,0,0,0,0,0,0,1,30,0,0,0,8,0)
		(0,1,0,0,0,27,0,0,0,0,0,1,0,0,0,0,0,0,0)
10	(4,46)	(0,0,0,0,0,2,1,0,0,0,25,0,0,12,0,1,0,0,9)
		(0,3,0,0,0,0,0,3,0,6,3,0,2,24,0,0,1,8,0)
		(0,0,0,4,0,2,0,0,1,0,1,0,3,28,3,0,0,8,0)
		(0,0,0,0,0,25,0,0,1,0,0,1,0,1,0,1,0,0,0)
11	(0,50)	(0,0,0,0,0,0,2,0,0,0,18,0,0,11,0,1,0,0,18)
		(0,1,0,0,0,0,0,5,0,9,3,0,4,19,0,0,0,9,0)
		(0,0,0,3,0,2,0,0,2,0,1,0,5,19,3,0,0,15,0)
		(0,0,0,0,0,16,0,0,2,0,0,0,0,1,1,2,0,0,0)
12	(2,48)	(1,0,0,0,0,0,2,0,0,0,15,1,0,6,0,1,0,1,23)
		(0,0,0,0,0,0,0,0,9,0,9,4,0,4,13,0,1,1,9,0)
		(0,0,0,1,0,2,0,0,4,0,2,0,4,18,2,0,0,16,1)
		(0,0,0,0,0,13,0,0,4,0,0,0,0,0,4,0,0,0,0)
13	(1,49)	(1,0,0,0,0,0,1,0,0,0,13,1,0,4,0,1,0,0,29)
		(0,0,0,1,0,1,0,7,1,9,1,0,6,9,0,3,1,10,1)
		(0,0,0,0,1,0,1,0,6,0,0,1,4,18,5,0,0,14,0)
		(0,0,0,0,0,12,0,0,3,0,0,0,0,0,5,0,0,0,0)
14	(1,49)	(0,1,0,1,0,0,0,1,0,0,8,3,0,8,0,1,1,1,25)
		(0,0,0,1,0,0,0,8,1,11,1,0,4,7,0,4,2,11,0)
		(0,0,0,0,0,0,0,0,4,1,0,0,9,15,6,1,0,14,0)
		(0,1,0,0,0,5,0,0,3,0,0,0,0,0,6,0,0,0,0)

A.3 Detailed results of experiment 3

Table A.9: *Histograms over the sample size in every generation. Column '#' shows the number of the generation. Column 'sample size' lists the histograms. The left-most value corresponds to the number of networks having the lowest sample size. The right-most value to the networks having the highest sample size.*

#	sample size
0	(6,6,4,6,5,8,2,7,6)
1	(7,5,0,4,6,10,7,8,3)
2	(6,8,1,6,2,8,4,6,9)
3	(3,6,2,7,1,7,5,11,8)
4	(5,8,1,11,1,2,8,8,6)
5	(3,7,0,18,1,2,7,7,5)
6	(2,8,0,19,0,1,7,6,7)
7	(0,7,2,25,3,1,4,5,3)
8	(0,4,1,26,4,3,3,5,4)
9	(0,3,4,24,6,1,2,6,4)
10	(0,1,6,22,0,3,1,13,4)
11	(0,1,8,13,1,3,1,18,5)
12	(0,2,8,15,1,3,1,16,4)
13	(1,3,6,21,2,4,0,11,2)
14	(1,0,6,24,2,6,0,10,1)

Table A.10: *Histograms over the settings for convolutional layer 1 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(14,20)	(10,13,11)	(19,15)	(19,15)	(4,8,6,10,6)
1	(11,14)	(8,12,5)	(12,13)	(19,6)	(1,8,4,7,5)
2	(7,16)	(5,12,6)	(10,13)	(16,7)	(3,4,4,7,5)
3	(8,18)	(7,12,7)	(12,14)	(20,6)	(0,6,7,6,7)
4	(2,31)	(9,22,2)	(9,24)	(20,13)	(0,4,11,2,16)
5	(2,43)	(3,38,4)	(8,37)	(18,27)	(1,3,20,5,16)
6	(4,46)	(3,44,3)	(9,41)	(23,27)	(0,2,22,6,20)
7	(7,43)	(1,46,3)	(8,42)	(24,26)	(1,1,21,3,24)

#	activation	pooling	size_x	size_y	masks
8	(6,44)	(3,43,4)	(5,45)	(22,28)	(0,1,13,5,31)
9	(5,45)	(4,45,1)	(7,43)	(28,22)	(1,1,17,4,27)
10	(6,44)	(4,46,0)	(7,43)	(25,25)	(0,3,10,7,30)
11	(2,48)	(5,45,0)	(7,43)	(26,24)	(0,1,17,9,23)
12	(1,49)	(5,44,1)	(10,40)	(25,25)	(0,0,18,8,24)
13	(3,47)	(2,47,1)	(15,35)	(17,33)	(2,2,8,9,29)
14	(3,47)	(3,46,1)	(12,38)	(14,36)	(0,1,5,14,30)

Table A.11: *Histograms over the settings for convolutional layer 2 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(6,14)	(10,7,3)	(11,9)	(8,12)	(3,3,5,2,7)
1	(0,15)	(11,3,1)	(8,7)	(11,4)	(0,1,3,2,9)
2	(0,13)	(4,7,2)	(6,7)	(9,4)	(0,0,4,3,6)
3	(0,12)	(1,11,0)	(11,1)	(12,0)	(0,0,2,0,10)
4	(1,8)	(2,7,0)	(6,3)	(8,1)	(0,0,2,0,7)
5	(0,6)	(1,5,0)	(1,5)	(6,0)	(0,0,5,0,1)
6	(0,7)	(0,7,0)	(2,5)	(7,0)	(0,0,7,0,0)
7	(0,3)	(0,3,0)	(1,2)	(3,0)	(0,0,3,0,0)
8	(0,2)	(0,2,0)	(0,2)	(2,0)	(0,0,2,0,0)
9	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
10	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
11	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
12	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
13	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
14	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)

Table A.12: *Histograms over the settings for the fully connected layer in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The layer column lists the histograms of the size in every possible layer.*

#	activation	layer
0	(28,22)	(6,1,3,1,1,0,5,0,4,4,2,2,1,1,2,2,3,2,0)
		(4,0,1,0,2,1,1,0,2,1,3,1,1,1,2,3,1,0,4)
		(1,0,1,0,0,2,0,2,1,1,1,0,0,1,3,0,1,3,1)
		(0,0,0,1,2,2,0,1,1,0,0,0,0,0,0,0,1,0,1)
1	(19,31)	(3,0,0,1,0,0,5,0,4,3,5,2,4,1,2,4,4,5,0)
		(2,0,0,0,8,1,1,0,4,2,1,1,1,2,1,6,2,0,5)
		(1,0,1,0,1,5,0,1,1,2,1,0,0,2,3,0,0,8,0)
		(0,0,0,1,4,3,0,1,4,0,0,0,0,0,0,0,4,0,4)
2	(14,36)	(1,0,0,0,1,0,5,0,4,7,6,2,2,1,1,5,6,4,0)
		(2,0,0,0,9,1,0,0,5,3,0,1,5,4,1,6,4,0,2)
		(0,0,3,1,2,5,1,1,0,3,1,0,0,4,5,0,0,6,1)
		(0,0,0,1,4,4,1,0,4,0,0,0,0,0,0,0,2,2,5)
3	(6,44)	(0,0,0,0,0,0,7,1,2,3,10,1,7,1,1,1,5,11,0)
		(1,0,0,0,11,2,0,0,8,0,0,2,3,3,0,8,5,0,6)
		(0,0,2,1,3,9,3,1,0,1,2,0,0,1,1,0,0,13,5)
		(0,0,0,2,7,1,1,1,8,1,0,0,0,0,0,1,7,3,7)
4	(1,49)	(1,0,0,0,0,0,11,0,0,3,5,1,7,1,0,0,7,14,0)
		(2,0,0,0,12,0,0,0,3,0,0,3,1,2,0,12,8,0,7)
		(0,0,3,0,3,4,6,1,0,2,0,0,0,1,1,0,1,14,6)
		(0,0,0,0,2,0,0,1,14,2,0,0,0,1,0,0,8,2,10)
5	(2,48)	(0,0,0,0,1,0,13,0,0,1,3,1,10,1,0,0,6,14,0)
		(2,0,0,1,12,0,0,1,4,1,0,1,1,0,0,11,6,0,10)
		(0,0,4,0,1,3,7,0,0,4,0,0,0,1,1,1,1,16,5)
		(0,0,0,0,3,1,0,1,9,2,1,0,0,1,0,0,11,2,12)
6	(1,49)	(0,0,0,0,1,0,8,0,1,2,3,5,8,2,0,0,5,15,0)
		(3,0,0,1,13,1,0,0,5,2,0,1,0,0,0,10,5,0,9)
		(0,0,6,0,0,4,7,0,0,1,0,0,0,2,0,4,0,16,6)
		(0,0,1,0,0,1,0,0,11,4,0,0,0,1,1,0,11,8,8)
7	(1,49)	(1,0,0,0,0,1,9,0,0,1,6,5,3,2,1,0,5,15,1)
		(3,0,0,1,13,1,0,0,5,2,0,1,0,0,0,11,3,0,10)
		(1,1,5,0,0,6,8,0,0,2,0,0,0,1,0,5,0,14,5)
		(0,0,1,0,1,2,0,0,9,6,0,0,0,2,2,0,8,9,8)
8	(2,48)	(0,0,0,0,1,0,3,0,1,0,7,8,4,2,2,0,2,18,2)
		(9,0,1,1,12,2,0,0,7,2,0,2,0,0,1,6,1,0,6)

#	activation	layer
		(2,0,5,0,0,6,3,0,0,0,1,0,0,2,0,8,0,15,8)
		(0,0,0,1,1,0,0,0,9,6,1,1,0,3,2,0,7,14,5)
9	(1,49)	(0,0,0,0,0,0,3,0,0,0,10,16,0,2,1,0,1,14,3)
		(6,0,3,1,11,1,1,0,14,0,0,1,1,0,0,3,0,0,8)
		(0,0,5,0,0,8,1,0,0,0,3,2,0,2,1,15,0,10,3)
		(0,0,0,1,2,0,0,0,8,6,4,1,1,1,0,0,2,21,3)
10	(0,50)	(0,0,0,0,0,0,3,0,1,0,9,18,0,1,0,0,1,15,2)
		(7,1,1,3,8,0,1,1,17,0,0,0,1,0,0,3,0,0,7)
		(1,0,4,0,0,8,1,0,0,0,2,1,0,1,5,13,0,13,1)
		(1,0,1,1,1,0,0,2,4,5,7,1,0,1,0,0,0,25,1)
11	(4,46)	(0,2,0,1,0,0,7,0,2,0,6,13,0,0,1,0,2,13,3)
		(7,1,3,2,8,0,3,0,14,2,0,0,1,0,0,4,0,0,5)
		(0,0,8,0,0,8,1,0,0,0,4,2,0,1,4,10,0,12,0)
		(2,0,1,3,2,2,0,1,5,3,4,0,0,0,0,1,0,24,2)
12	(2,48)	(0,1,0,2,1,1,6,0,3,0,5,10,0,0,0,0,3,16,2)
		(13,0,6,2,4,0,4,1,11,1,0,1,2,1,0,1,0,0,3)
		(0,0,7,0,0,6,0,0,0,0,5,3,0,1,1,12,0,15,0)
		(1,0,1,1,2,1,0,1,3,3,3,0,0,0,0,0,0,30,4)
13	(2,48)	(1,1,0,1,0,2,2,1,1,0,5,15,0,0,0,0,9,12,0)
		(6,0,3,2,0,0,6,0,14,1,0,1,7,1,0,2,1,0,6)
		(0,0,4,0,0,5,0,0,0,0,0,6,0,3,2,13,1,16,0)
		(1,0,0,2,1,2,0,0,0,2,9,0,0,0,0,0,0,33,0)
14	(3,47)	(0,0,0,1,0,0,1,0,0,0,7,12,0,0,0,1,20,8,0)
		(2,0,6,2,0,0,5,0,16,0,1,2,9,0,2,1,0,0,4)
		(0,0,4,0,0,9,0,0,0,0,0,8,0,4,0,16,1,8,0)
		(0,0,0,5,1,4,1,0,0,3,7,0,0,0,0,0,0,29,0)

A.4 Detailed results of experiment 4

Table A.13: *Histograms over the sample size in every generation. Column '#' shows the number of the generation. Column 'sample size' lists the histograms. The left-most value corresponds to the number of networks having the lowest sample size. The right-most value to the networks having the highest sample size.*

#	sample size
0	(5,5,6,3,3,10,5,4,9)
1	(8,3,7,0,6,5,8,1,12)

#	sample size
2	(6,7,10,0,5,1,6,4,11)
3	(3,10,5,1,7,2,8,3,11)
4	(3,8,8,4,4,2,6,4,11)
5	(2,5,12,5,6,5,5,2,8)
6	(2,7,12,9,1,5,8,1,5)
7	(0,7,17,11,0,2,7,0,6)
8	(0,12,14,9,1,4,9,1,0)
9	(0,10,14,9,1,5,8,3,0)
10	(1,11,10,7,0,4,14,3,0)
11	(1,8,8,11,0,8,10,4,0)
12	(1,7,7,11,0,9,12,2,1)
13	(1,6,11,8,0,7,11,5,1)
14	(0,2,4,15,1,7,8,13,0)

Table A.14: *Histograms over the settings for convolutional layer 1 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(18,22)	(14,13,13)	(20,20)	(20,20)	(9,9,9,6,7)
1	(11,24)	(13,8,14)	(17,18)	(20,15)	(7,8,7,4,9)
2	(11,29)	(18,13,9)	(16,24)	(19,21)	(5,6,12,7,10)
3	(7,31)	(14,8,16)	(14,24)	(21,17)	(1,4,9,6,18)
4	(10,27)	(15,10,12)	(11,26)	(21,16)	(1,2,11,5,18)
5	(9,29)	(14,12,12)	(12,26)	(21,17)	(0,1,10,4,23)
6	(9,30)	(15,14,10)	(13,26)	(20,19)	(0,0,11,1,27)
7	(6,37)	(20,13,10)	(8,35)	(27,16)	(1,1,8,0,33)
8	(9,38)	(17,25,5)	(8,39)	(29,18)	(0,0,11,1,35)
9	(13,36)	(13,32,4)	(6,43)	(19,30)	(2,0,9,1,37)
10	(18,30)	(11,34,3)	(9,39)	(17,31)	(2,0,8,1,37)
11	(12,37)	(12,35,2)	(7,42)	(21,28)	(1,1,10,1,36)
12	(11,39)	(17,31,2)	(9,41)	(24,26)	(1,1,11,3,34)
13	(2,48)	(18,27,5)	(9,41)	(23,27)	(1,1,5,2,41)
14	(5,45)	(21,27,2)	(13,37)	(24,26)	(2,0,6,4,38)

Table A.15: *Histograms over the settings for convolutional layer 2 in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The histogram values in the 'pooling' column correspond to the pooling functions in the order (none, max, avg). Columns 'size_x' and 'size_y' show the size of the convolution kernels in each direction, the left value corresponds to size 2, the right to size 3. The last column 'masks' lists the histograms over the number of convolution kernels in this layer.*

#	activation	pooling	size_x	size_y	masks
0	(7,9)	(7,6,3)	(9,7)	(8,8)	(1,4,6,3,2)
1	(3,9)	(5,6,1)	(5,7)	(7,5)	(0,3,4,1,4)
2	(3,8)	(6,4,1)	(7,4)	(3,8)	(2,0,6,1,2)
3	(1,6)	(2,5,0)	(4,3)	(2,5)	(0,0,4,0,3)
4	(2,2)	(2,1,1)	(3,1)	(1,3)	(0,0,3,0,1)
5	(0,1)	(0,1,0)	(0,1)	(0,1)	(0,0,0,0,1)
6	(0,3)	(0,3,0)	(1,2)	(0,3)	(0,0,0,0,3)
7	(0,2)	(0,2,0)	(0,2)	(0,2)	(0,0,0,0,2)
8	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
9	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
10	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
11	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
12	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
13	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)
14	(0,0)	(0,0,0)	(0,0)	(0,0)	(0,0,0,0,0)

Table A.16: *Histograms over the settings for the fully connected layer in every generation. Column '#' shows the number of the generation. The left values in the 'activation' column corresponds to the tanh activation, the right one to 'relu'. The layer column lists the histograms of the size in every possible layer.*

#	activation	layer
0	(28,22)	(6,1,3,1,1,0,5,0,4,4,2,2,1,1,2,2,3,2,0) (4,0,1,0,2,1,1,0,2,1,3,1,1,1,2,3,1,0,4) (1,0,1,0,0,2,0,2,1,1,1,0,0,1,3,0,1,3,1) (0,0,0,1,2,2,0,1,1,0,0,0,0,0,0,0,1,0,1)
1	(19,31)	(3,0,0,1,0,0,5,0,4,3,5,2,4,1,2,4,4,5,0) (2,0,0,0,8,1,1,0,4,2,1,1,1,2,1,6,2,0,5) (1,0,1,0,1,5,0,1,1,2,1,0,0,2,3,0,0,8,0) (0,0,0,1,4,3,0,1,4,0,0,0,0,0,0,0,4,0,4)

#	activation	layer
2	(14,36)	(1,0,0,0,1,0,5,0,4,7,6,2,2,1,1,5,6,4,0)
		(2,0,0,0,9,1,0,0,5,3,0,1,5,4,1,6,4,0,2)
		(0,0,3,1,2,5,1,1,0,3,1,0,0,4,5,0,0,6,1)
		(0,0,0,1,4,4,1,0,4,0,0,0,0,0,0,0,2,2,5)
3	(6,44)	(0,0,0,0,0,0,7,1,2,3,10,1,7,1,1,1,5,11,0)
		(1,0,0,0,11,2,0,0,8,0,0,2,3,3,0,8,5,0,6)
		(0,0,2,1,3,9,3,1,0,1,2,0,0,1,1,0,0,13,5)
		(0,0,0,2,7,1,1,1,8,1,0,0,0,0,0,1,7,3,7)
4	(1,49)	(1,0,0,0,0,0,11,0,0,3,5,1,7,1,0,0,7,14,0)
		(2,0,0,0,12,0,0,0,3,0,0,3,1,2,0,12,8,0,7)
		(0,0,3,0,3,4,6,1,0,2,0,0,0,1,1,0,1,14,6)
		(0,0,0,0,2,0,0,1,14,2,0,0,0,1,0,0,8,2,10)
5	(2,48)	(0,0,0,0,1,0,13,0,0,1,3,1,10,1,0,0,6,14,0)
		(2,0,0,1,12,0,0,1,4,1,0,1,1,0,0,11,6,0,10)
		(0,0,4,0,1,3,7,0,0,4,0,0,0,1,1,1,1,16,5)
		(0,0,0,0,3,1,0,1,9,2,1,0,0,1,0,0,11,2,12)
6	(1,49)	(0,0,0,0,1,0,8,0,1,2,3,5,8,2,0,0,5,15,0)
		(3,0,0,1,13,1,0,0,5,2,0,1,0,0,0,10,5,0,9)
		(0,0,6,0,0,4,7,0,0,1,0,0,0,2,0,4,0,16,6)
		(0,0,1,0,0,1,0,0,11,4,0,0,0,1,1,0,11,8,8)
7	(1,49)	(1,0,0,0,0,1,9,0,0,1,6,5,3,2,1,0,5,15,1)
		(3,0,0,1,13,1,0,0,5,2,0,1,0,0,0,11,3,0,10)
		(1,1,5,0,0,6,8,0,0,2,0,0,0,1,0,5,0,14,5)
		(0,0,1,0,1,2,0,0,9,6,0,0,0,2,2,0,8,9,8)
8	(2,48)	(0,0,0,0,1,0,3,0,1,0,7,8,4,2,2,0,2,18,2)
		(9,0,1,1,12,2,0,0,7,2,0,2,0,0,1,6,1,0,6)
		(2,0,5,0,0,6,3,0,0,0,1,0,0,2,0,8,0,15,8)
		(0,0,0,1,1,0,0,0,9,6,1,1,0,3,2,0,7,14,5)
9	(1,49)	(0,0,0,0,0,0,3,0,0,0,10,16,0,2,1,0,1,14,3)
		(6,0,3,1,11,1,1,0,14,0,0,1,1,0,0,3,0,0,8)
		(0,0,5,0,0,8,1,0,0,0,3,2,0,2,1,15,0,10,3)
		(0,0,0,1,2,0,0,0,8,6,4,1,1,1,0,0,2,21,3)
10	(0,50)	(0,0,0,0,0,0,3,0,1,0,9,18,0,1,0,0,1,15,2)
		(7,1,1,3,8,0,1,1,17,0,0,0,1,0,0,3,0,0,7)
		(1,0,4,0,0,8,1,0,0,0,2,1,0,1,5,13,0,13,1)
		(1,0,1,1,1,0,0,2,4,5,7,1,0,1,0,0,0,25,1)
11	(4,46)	(0,2,0,1,0,0,7,0,2,0,6,13,0,0,1,0,2,13,3)
		(7,1,3,2,8,0,3,0,14,2,0,0,1,0,0,4,0,0,5)
		(0,0,8,0,0,8,1,0,0,0,4,2,0,1,4,10,0,12,0)

#	activation	layer
		(2,0,1,3,2,2,0,1,5,3,4,0,0,0,0,1,0,24,2)
12	(2,48)	(0,1,0,2,1,1,6,0,3,0,5,10,0,0,0,0,0,3,16,2)
		(13,0,6,2,4,0,4,1,11,1,0,1,2,1,0,1,0,0,3)
		(0,0,7,0,0,6,0,0,0,0,5,3,0,1,1,12,0,15,0)
		(1,0,1,1,2,1,0,1,3,3,3,0,0,0,0,0,0,30,4)
13	(2,48)	(1,1,0,1,0,2,2,1,1,0,5,15,0,0,0,0,9,12,0)
		(6,0,3,2,0,0,6,0,14,1,0,1,7,1,0,2,1,0,6)
		(0,0,4,0,0,5,0,0,0,0,0,6,0,3,2,13,1,16,0)
		(1,0,0,2,1,2,0,0,0,2,9,0,0,0,0,0,0,33,0)
14	(3,47)	(0,0,0,1,0,0,1,0,0,0,7,12,0,0,0,1,20,8,0)
		(2,0,6,2,0,0,5,0,16,0,1,2,9,0,2,1,0,0,4)
		(0,0,4,0,0,9,0,0,0,0,0,8,0,4,0,16,1,8,0)
		(0,0,0,5,1,4,1,0,0,3,7,0,0,0,0,0,0,29,0)