Technische Universität Hamburg-Harburg
Vision Systems

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

A Comparison and Evaluation of Neural Network-based Classification Approaches for the Purpose of a Robot Detection on the Nao Robotic System

Project Thesis

Chris Kahlefendt

April 05, 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, April 05, 2017 Chris Kahlefendt

Contents

List of Figures

List of Tables

List of Acronyms

API Application Programming Interface

CNN Convolutional Neural Network

FFT Fast Fourier Transform

FP False Positive

FPR False Positive Rate

FN False Negative

HULKs Hamburg Ultra Legendary Kickers

ILSVRC ImageNet Large Scale Visual Recognition Competition

JSON Javascript Object Notation

NN Neural Network

PSD Power Spectral Density

ReLU Rectified Linear Unit

SPL Standard Platform League

SSE Streaming SIMD Extensions

TUHH Technische Universität Hamburg (University of Technology Hamburg)

TP True Positive

TPR True Positive Rate

TN True Negative

1 Introduction

This thesis is written as part of the research the robot soccer team Hamburg Ultra Legendary Kickers (HULKs) does at the TUHH. The following chapter will introduce the used robotic platform "NAO", give a short overview about the RoboCup, explain the motivation and goals of this thesis, present some existing solutions for robot detections and explain how this thesis is going to approach the problem.

1.1 The NAO Robot

The NAO is a robot developed by the French company *Aldebaran Robotics* which is now part of the Japanese *Softbank Robotics Corp.*. It has been in development since 2006 [23] and is currently available in its fifth version. This is the robot that is being used in the Standard Platform League (SPL) and is also the robot used for this thesis.

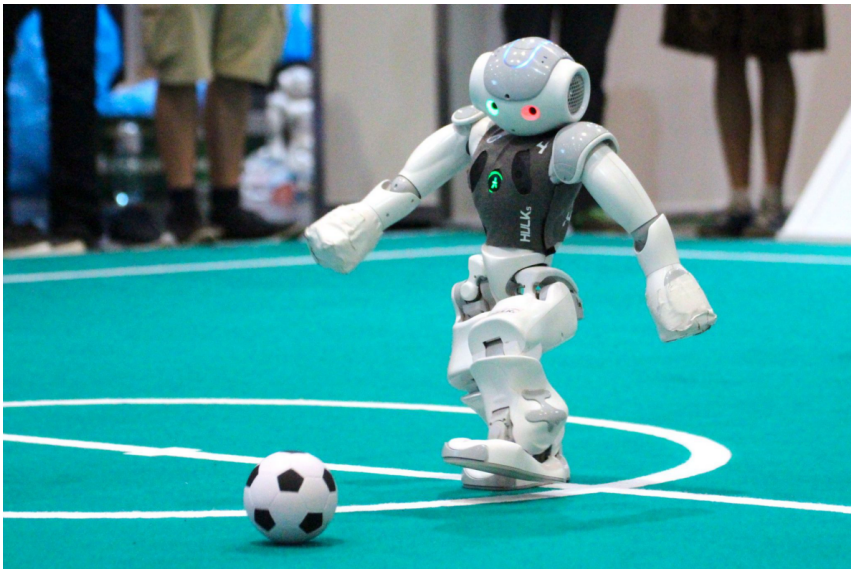


Figure 1.1: NAO robot kicking a ball at RoboCup 2016 in Leipzig [16]

The NAO is powered by an Intel ATOM Z530 processor with a clock speed of 1.6 GHz and 1 GB RAM. It has two front facing cameras, one in the forehead and one at the position of it's mouth. As visible in figure 1.2 the perceptional fields of the two cameras only overlap partially. Therefore stereo vision is not possible. Both cameras support resolutions of up to 1280x960 px with 29-30 frames per second [24]. The HULKs software framework uses a resolution of 640x480 px and the YCbCr[1] color space for it's computations.

[1]Details about this color space can be found in [14], [21] and [22].

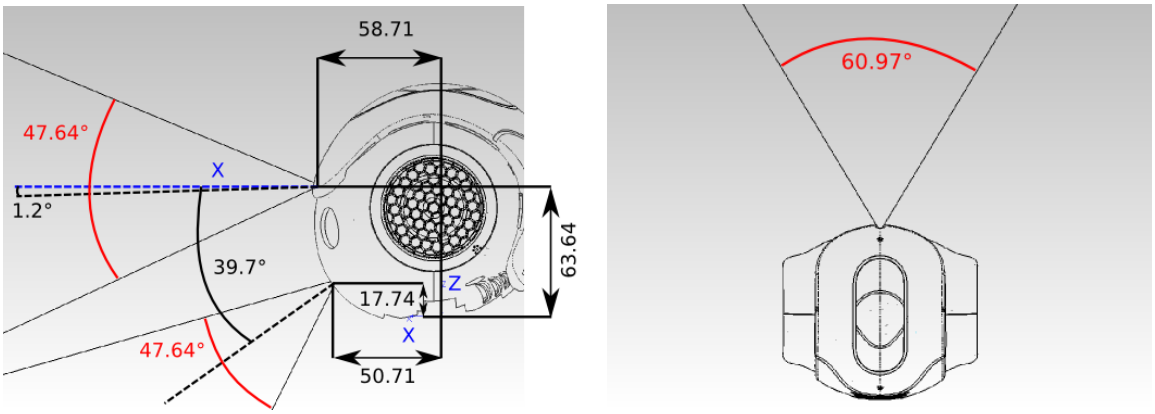


Figure 1.2: Position and perceptional field of the cameras used in the NAO robot[1]

1.2 The RoboCup

RoboCup is the short form for “Robot World Cup Initiative” and is an annual event where teams from all over the world compete in various activities using different kinds of robots. The general idea of the RoboCup is not just to compete and play soccer with robots but rather to provide a platform where a “wide range of technologies especially concerning multi-agent research can be integrated and examined” [13]. After it’s conception in 1995 the first RoboCup was held in Nagoya, Japan in 1997 and has since been carried out in over 15 different countries [17].

Standard Platform League

The SPL is the league the HULKs participate in. In the SPL every team has to use the NAO robotic system and is not allowed to modify it in any way. This removes the necessity of having to cope with hardware problems and sets the focus on the software used.

A SPL game takes twenty minutes and is played in two halves of ten minutes each, with a short break in between. During a game the robots have to act completely autonomous and do not receive any commands from outside. This means the robots have to be able to localize themselves on the field, find the ball, avoid other robots, find the enemy’s goal and score all by themselves.

By gradually implementing new rules the RoboCup officials try to increase the difficulty of the competition with each year. This way the researchers working in the teams often have to cope with unfamiliar situations and need to find new ways to deal with them. One of these changes was switching from a completely red to a black and white ball for the 2016 RoboCup. For the 2017 RoboCup it is planned to switch from flat green carpet to artificial turf for the soccer field. The lighting conditions are due to change as well. Instead of using only artificial light sources the new rules intend to put every field close to a window to force natural lighting conditions [18].

1.3 Motivation

Being able to perceive other players is crucial for playing soccer. It opens the possibility to plan ahead and allows for creating strategies and therefore secures a better chance of winning. This not only counts for humans playing soccer, it is also very important for robot soccer.

A robot which is not able to see other players will:

- be much more likely to loose the ball to the enemy when dribbling.
- tend to run into other robots where it might either fall and loose time or cause a pushing foul which will bring a penalty of at least 45 seconds with it[18].
- not be able to tell where the enemy goalkeeper is positioned inside the goal and might shot straight at it.

When a robot is able to see other players it allows for:

- better ways of path planning by avoiding other robots on the way to the ball and finding the optimal way to the goal.
- using the information to find a spot with a lot of space so the robot is free for passes.
- using information about other robots to better localize the robot itself.
- informing their teammates about enemies that are following, so they can act accordingly.

So far the HULKs software framework does not contain any means of detecting other robots, while other teams have already implemented various ways of realizing this. The absence of a robot detection creates a major disadvantage for the HULKs, which is why the motivation for this thesis was to catch up on this part.

1.4 Overview of existing Approaches

This section will shortly present the means tested and implemented in works of other teams. Works of the following teams were examined: *rUNSWift*, *Nao-Team HTWK* and *B-Human*.

rUNSWift

There are two works describing the techniques tested by the two times SPL world champion from Sydney, Australia. The team successfully tested a decision tree classifier which was able to achieve up to 97.7% accuracy and a false positive rate (FPR)[2] of about 2%[6].

[2] An explanation of the terms accuracy and FPR is given in 3.1

That is a very good result but according to [4] it was not implemented on the NAO "due to lack of processing power to run the algorithm". A Bayesian classifier which achieved between 80-87% accuracy and a FPR of 9% was proposed as a replacement [4].

Nao-Team HTWK

The NAO-Team HTWK implemented a feature-based linear regression approach in 2012. This method is documented in [11] and achieved about 90% accuracy, 87% recall and 3% FPR.

B-Human

B-Human uses an approach that does not rely on a trained classifier. According to [3] it performs the robot detection in less than two milliseconds. It basically scans for areas inside the field boundaries which do not contain field color and defines them as obstacles. If a jersey color is found it will become either a teammate or an enemy robot.

The approach used by B-Human requires good knowledge of where the soccer field starts and ends and the algorithm needs to have prior knowledge about the proposed objects to make sure only actual robots are detected. Since for the HULKs software framework this was not a given at the time of creating this thesis, the classification approach used by the other teams was a more suitable concept.

1.5 Goals

The aim of this thesis is to examine different Neural Network (NN) architectures for the purpose of a robot detection which is implementable on the NAO according to the set requirements explained in 4.2. Especially Convolutional Neural Networks (CNNs) which have become very famous in the field of object recognition over the recent years are a very interesting option and have not been evaluated in this context yet. Since the approach of using a linear regression, which is comparable to using a simple NN, showed very promising results[11]. A comparison between feature-based NN classification and CNNs is also part of this work.

1.6 Problem Overview and Thesis Structure

A common way of approaching an object detection is to separate it into two parts. The first part, the object localization, describes the part of the algorithm that looks for areas containing objects. The second part, the object classification, uses a trained classifier to calculate which of the labels it knows fits best to the areas provided by the localization. An example for this can be seen in figure 1.3.

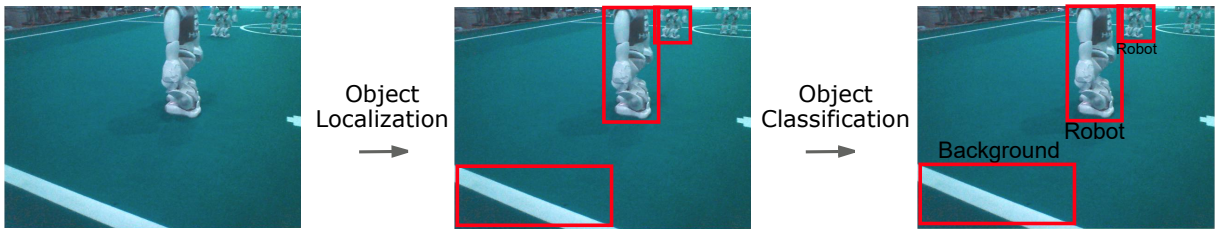


Figure 1.3: Common steps of an object detection

This separation into two parts was also done during this thesis. First an algorithm that is able to generate a set of robot candidate areas was written. This candidate generation algorithm is explained in-depth in chapter 2.

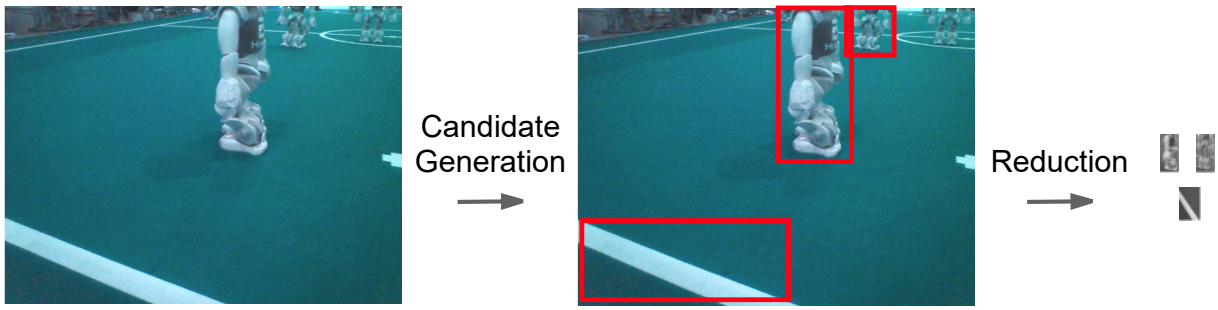


Figure 1.4: Steps of the object localization used for this thesis

As can be seen in figure 1.4 the areas defined by the candidate generation are reduced to smaller sizes and the color information is discarded. Why and how this was done is explained in chapter 3 along with the theoretical basics needed to understand how the classification works. Figure 1.5 shows the classification approaches that were examined.

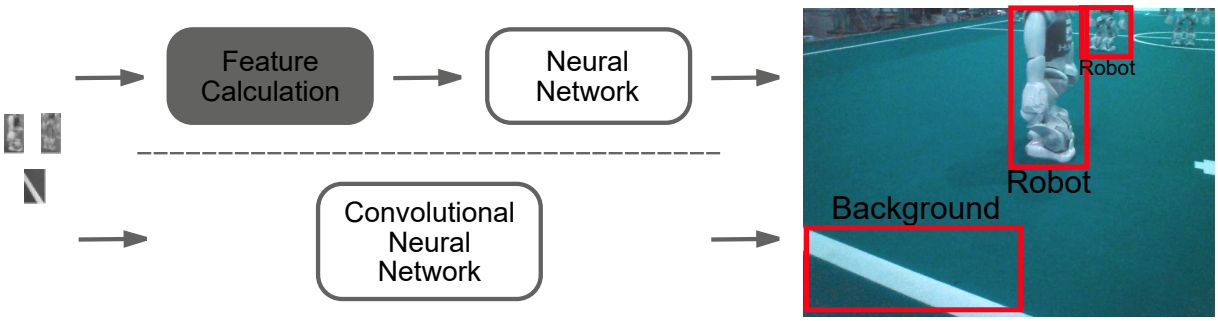


Figure 1.5: Steps of the object classification used for this thesis

Using the previously generated candidates both a feature-based NN method and CNNs were tested and evaluated. The results of this evaluation can be found in chapter 4.

Chapter 5 concludes this thesis by providing a summary of the results and presenting suggestions on how to proceed on this topic.

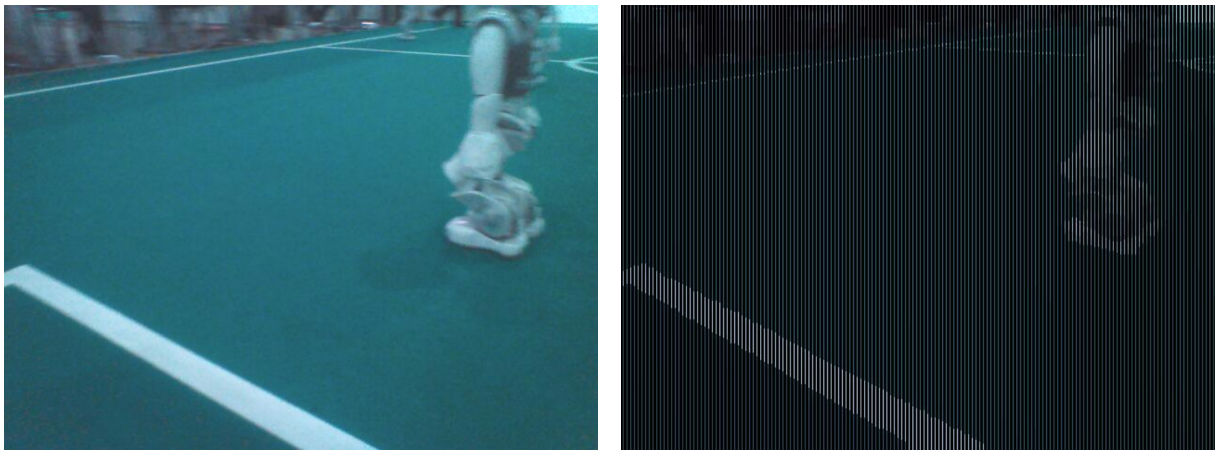
2 Candidate Generation

In order to recognize whether an image contains a robot the first step that has to be taken, is to find image areas which are have a high chance of containing a robot. These areas, which will be referred to as robot candidates, can then be used as an input for a classifier that, based on its training, decides whether a candidate is a robot or not. The following chapter will explain the algorithm used for extracting these candidates and present it's runtime and results.

2.1 Cluster Creation

Since robot soccer is played on a soccer field which consists of different shades of green, the color information can be used to define what could be interesting areas for finding robots. In general anything the robot sees inside the soccer field that differs from the green field color can be assumed to be some kind of object. During a game there are only a few different objects on the field the robot can see: the ball, other robots and sometimes a referee's leg. Therefore the idea of finding robot candidates that is used for this thesis is to look for areas that consist of shapes which do not contain field color.

The HULKs software framework provides a very helpful tool for analyzing images, called the Image Segmenter. The Image Segmenter creates a scanline for every fourth pixel column in an image. This means, that it moves through every fourth pixel column and calculates it's local gradients in vertical direction. For each of the scanned columns it then defines one dimensional regions based on the calculated gradients. Start and end of these regions are marked by gradients exceeding a set threshold. The Field Color Detection then marks the regions which are detected as field color and passes them on to the algorithm explained here. Figure 2.1 shows an original image and it's segmented image.



(a) Original image (b) Segmented image

Figure 2.1: Image scanned by Image Segmenter

The Image Segmenter creates regions of comparable gradient size which are shown as lines

of the same color. Due to only every fourth column being scanned the segmented image appears very dark and it is hard to see the single regions. For visualization purposes figure 2.2 fills these blank spaces between the scanlines with neighboring regions.

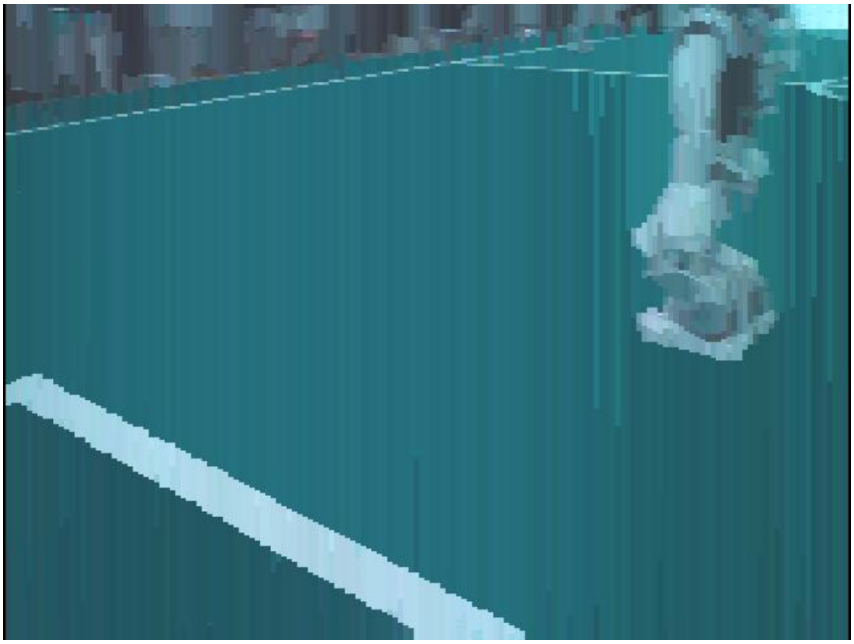


Figure 2.2: Segmented image with filled blank spaces

To be able to specify an actual area of interest the one dimensional regions need to be combined to two-dimensional clusters. The way the algorithm does this is as follows:

Merging of single line regions:

First the algorithm moves through every single region on a scanline and merges it with the previous region if neither contains field color. Figure 2.3 shows the merging in progress.

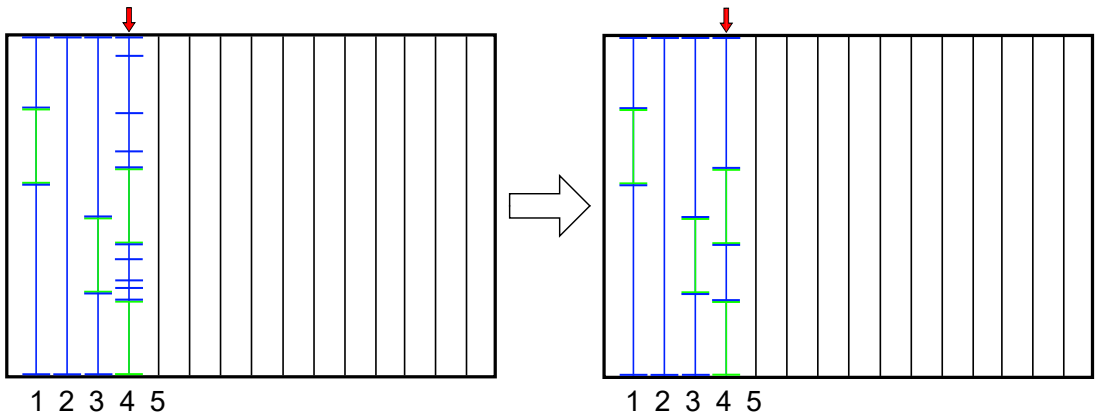


Figure 2.3: Merging of one dimensional image regions. Green lines represent field color regions, blue lines are non-field color regions and black lines have not been evaluated yet.

As can be seen by the arrow on top of the image the algorithm currently checks scanline number four. Before merging, many small, separated, non-field color regions exist. After merging, the blue regions are only separated by green field-color regions.

Cluster Forming:

Now for each non field color region on the current scanline it is checked whether it can be combined with any non field color cluster on the previous scanline. Clustering is allowed when:

1. the two regions overlap,
2. the region is longer than k_{min_length},
3. the region is not more than k_{grow} times larger than the previous,
4. the region is not more than k_{shrink} times smaller than the previous.

Where k_{min_length}, k_{grow} and k_{shrink} are tuneable parameters. When all of the four criteria are met the region will become part of the previous cluster.

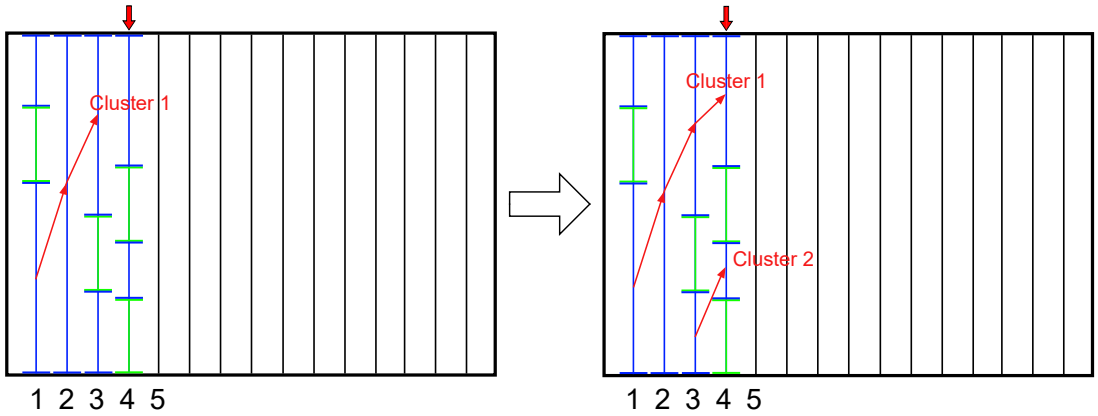


Figure 2.4: Creation of two dimensional clusters. The red arrows show how the existing clusters change during step four.

As can be seen the upper blue region in scanline four will be combined with the existing cluster symbolized by the red arrows. The lower blue region cannot be combined with the existing cluster because they do not overlap. Instead it forms a new cluster with the blue region at the bottom of scanline three.

Creating bounding boxes:

After creating the clusters, bounding boxes are formed from the minimum and maximum x and y values in the regions forming the clusters. A possible result of the clustering algorithm can be seen in figure 2.5.

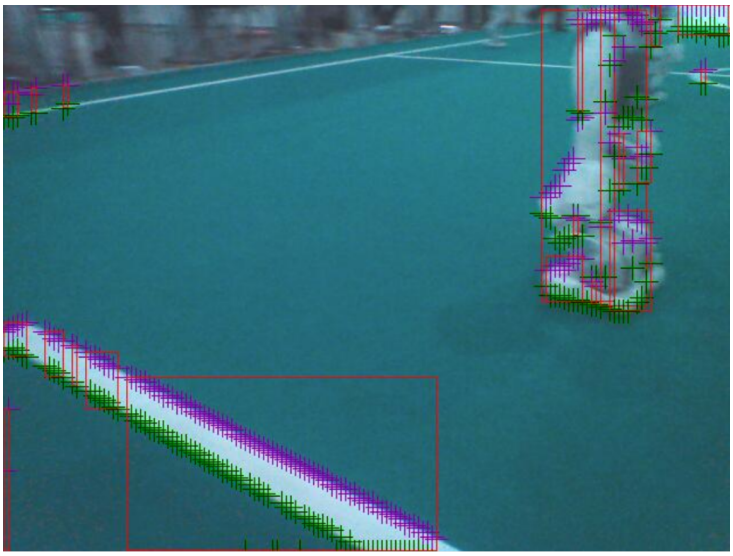


Figure 2.5: Clustering result. The calculated bounding boxes are shown in red. The purple and green crosses represent beginning and end of the used regions inside a cluster.

Looking at figure 2.5 it is obvious that the algorithm creates a large number of areas not containing robots. One reason for this is the used field color detection algorithm demonstrated in figure 2.6.

Figure 2.6: Field color detection image with all field color pixels marked in purple.

It is easy to see that there are a lot of pixels in the robot which were falsely classified as field color. This is also the reason for the small clusters on the right side of the robot in figure 2.5. The clusters on the top and bottom left can be explained by the sensitivity of the field color detection to changes in lighting conditions. Since some of the pixels in these regions become very dark, the algorithm does not consider them as field color anymore and the clustering program starts to create clusters from these areas. Another very obvious problem are field lines, as is noticeable in the lower left corner of figure 2.5.

2.2 Cluster Evaluation

Evaluation of all created clusters would cause extreme computational costs, therefore different measures for filtering had to be taken. Using the following three criteria for filtering proved to reduce the amount of irrelevant clusters to a minimum.

- Overall cluster size
- Cluster aspect ratio
- Variance of region lengths inside clusters

Cluster Size Filter

This filter is very straightforward. By constraining the minimum width w_{min} and height h_{min} a cluster needs to have and the maximum width w_{max} and height h_{max} it can possibly reach to still be considered a robot candidate, a lot of the generated clusters can be eliminated. The application of the filter can be seen in figure 2.7. The white boxes indicate clusters that are removed by the size filter. The figure also demonstrates one of the problems with this filter. Sometimes it can remove clusters which do contain robots like the one in the background. Since it is more important to detect robots which are close and allowing smaller clusters tends to also allow more non-robot clusters, this is a compromise that had to be taken.

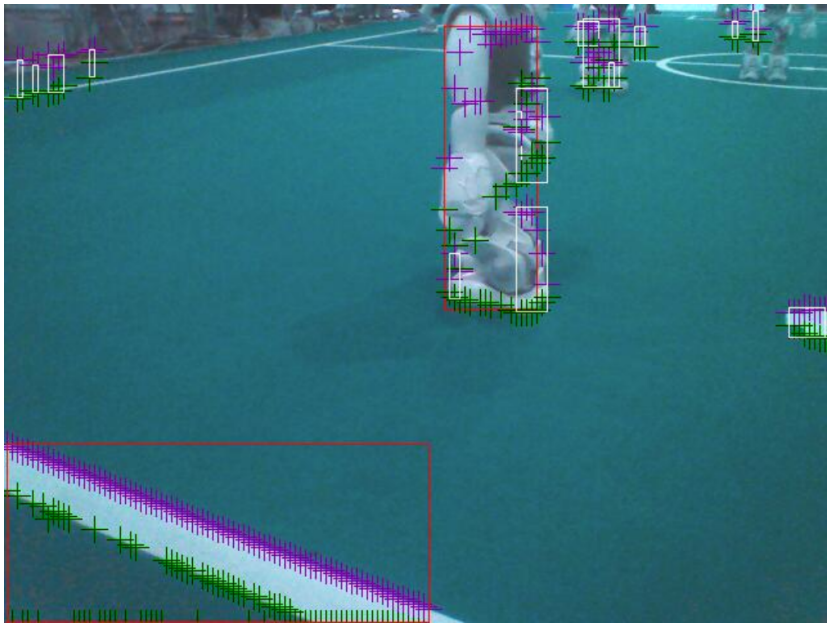


Figure 2.7: Application of size filter

Cluster Aspect Ratio Filter

The second filter applied takes the aspect ratio of a robot into account. The way it was implemented it assumes a robot cluster to be taller than wide. This makes sure that clusters which are wider than tall are not considered to be robot candidates. The effect is demonstrated in figure 2.8 where black bounding boxes resemble the clusters which are discarded. How the filter acts can be controlled via the two parameters w_{tol} and h_{tol}.

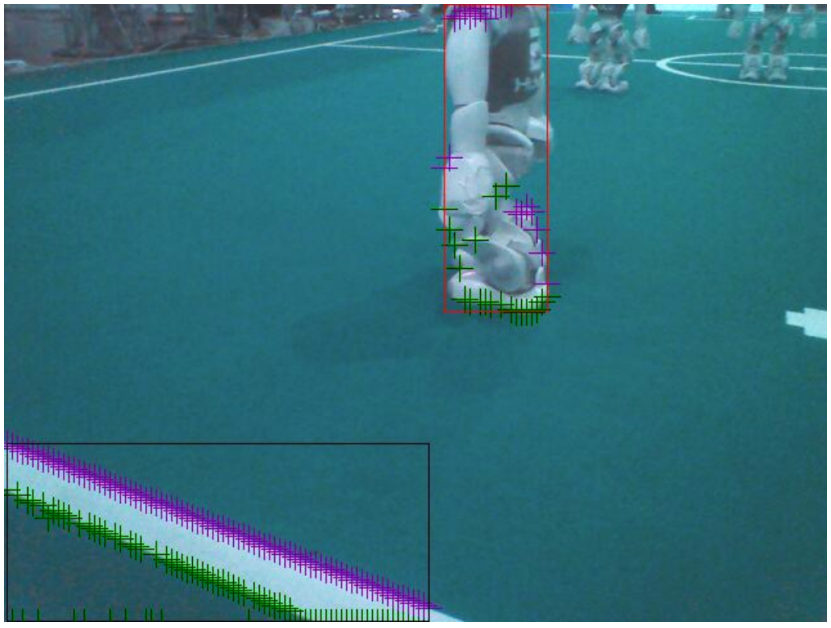


Figure 2.8: Application of aspect ratio filter

Variance of Region Lengths Filter

Applying the first two filters already managed to exclude many non-robot clusters. One problem that remained was clusters which formed on field lines because they often reached the same sizes as the robot clusters. Figure 2.9 represent a situation where this happens.

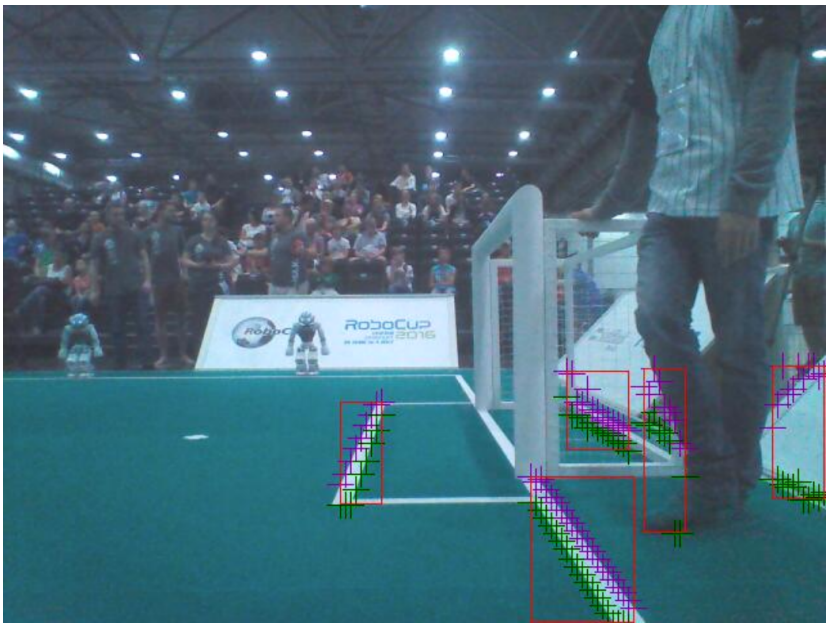


Figure 2.9: Problematic clusters after size and ratio filter

As can be seen in figures 2.9 and 2.8 the clusters on field lines share a common characteristic. The lengths of the regions inside the cluster are almost constant. By calculating the variance of these lengths this characteristic can be used to filter them. The formula for variance is as follows:

$$V(X) = \frac{\sum_{i=1}^{n} x_i^2}{N} - \mu^2 \qquad \forall\ x_i\ \epsilon\ [x_1, x_2, x_3, ..., x_n] \tag{2.1}$$

x_i : a single region length
μ : mean of all region lengths
N : total amount of regions in cluster

While testing, this definition for variance proved to be problematic. Due to it's results differing a lot depending on the lengths of the regions inside the cluster, it was not possible to find a suitable threshold for the variance value. When a cluster is formed close to the camera the lengths get larger whereas clusters further away hold shorter length regions. This effect can be witnessed when comparing the field line clusters in figure 2.9 and 2.8. To account for this, the variance was normed by the mean of all region lengths.

$$V_n(X) = \frac{V(X)}{\mu} \tag{2.2}$$

The filter's results on the clusters from figure 2.9 are shown in figure 2.10. Here the eliminated clusters are marked in blue.

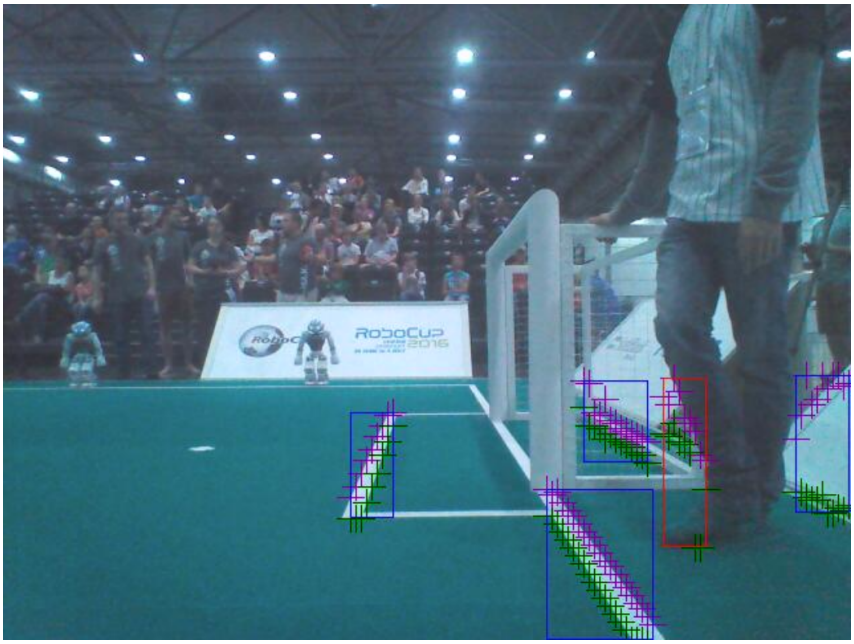


Figure 2.10: Application of variance filter

Since the clusters which do contain robots generally consist of regions of varying lengths, they are not affected by this filter. Figure 2.11 provides two images showing the final results after applying all three filters.

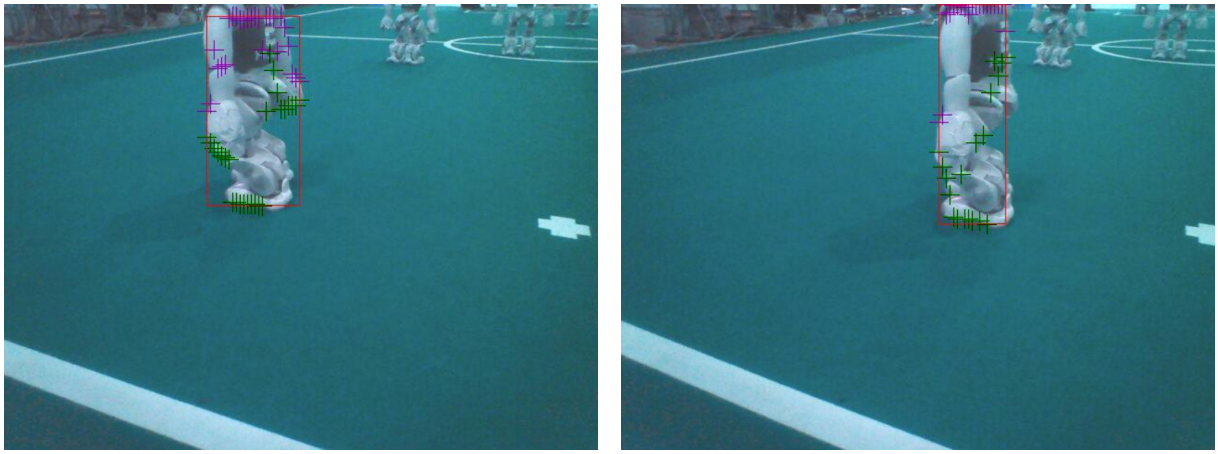


Figure 2.11: Final results after applying all three filters

All the parameters which were mentioned in this section can be controlled through a configuration file and are tunable while the robot is running the program. The configuration file used for this thesis can be found at A.1.

2.3 Performance

In order to evaluate the performance of the candidate generation algorithm it was run on 14109 different pictures which had been recorded during numerous Robocup games. The following section will analyze how the algorithm performed in terms of runtime and produced candidates.

During testing on said 14109 pictures the algorithm achieved the following results:

Candidate Statistics

- Overall amount of candidates generated: 4302
- Average amount of candidates per image: 0.305
- Maximum amount of candidates on a single image: 7
- Amount of candidates being robots: 1641 (38.15%)
- Amount of candidates being background: 2661 (61.85%)

These numbers show that even after applying all three filters the algorithm still produces a large number of background candidates. This also demonstrates the necessity for a classifier for a reliable robot detection.

Runtime statistics

For runtime evaluation the runtime for each part of the algorithm was tested separately. The algorithm was split in the following parts:

- Cluster creation
- Bounding box calculation
- Filter application
- Cropping and resizing
- Conversion for classifier

Cluster creation, bounding box calculation and filter application are independent of the amount of robot candidates and have to be run on every image. Cropping, resizing and the conversion for the classifier on the other hand are only run when robot candidates were found.
Cropping, resizing and the conversion for classifier parts are necessary to reshape the data in a form which is usable by the used C++ libraries. Cropping and resizing describes the algorithm which is needed to reduce the candidate frame to the target size. Conversion then scales all it's values to values between 0 and 1.

For measuring the runtime the algorithm was run on the NAO robot with a exemplary target image size of 17x27 pixels[3]. Overall the results of 4800 cycles were recorded. The results of the measurements can be examined in figures 2.12 and 2.13. The boxplot definition used in this thesis is the one defined by John W. Tukey [27].

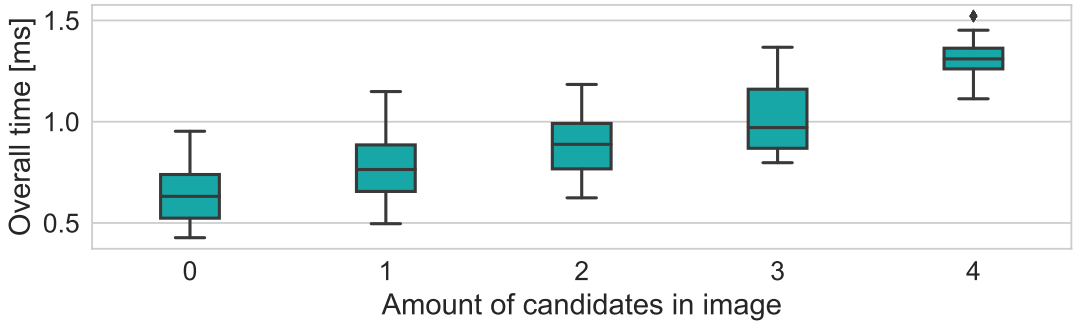


Figure 2.12: Overall candidate generation times

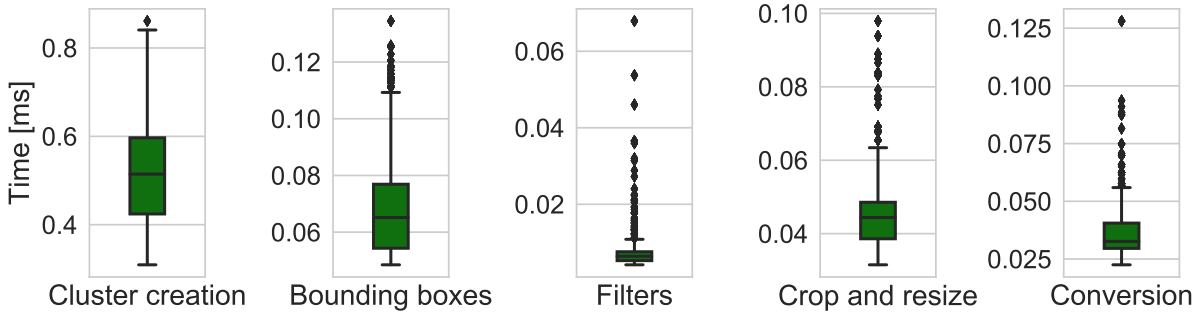


Figure 2.13: Runtimes per candidate generation part

Figure 2.12 reflects the fact that cluster creation, bounding boxes calculation and filter application have to be done on every image, even if it will not yield any robot candidates. The algorithm idles at an average of 0.63 ms on images without candidates, the maximum measured time being 0.95 ms. With increasing amount of candidates per image these times increase by the times needed for cropping and resizing and the conversion for the classifier. Cropping and resizing needs an average of 0.045 ms per candidate with a maximum measured at 0.098 ms. The conversion takes 0.037 ms on average and it's maximum was measured at 0.128 ms per candidate.

[3]The image size that is going to be fed to the classifier, here 17 pixels wide and 27 pixels high. How these were determined is explained in 3.3

A linear approximation following these results looks like this:

$$t_{avg} = 0.63\ ms + (0.045\ ms + 0.037\ ms) \cdot n \tag{2.3}$$

$$t_{max} = 0.95\ ms + (0.098\ ms + 0.128\ ms) \cdot n \tag{2.4}$$

t_{avg} : average time per candidate
t_{max} : maximum time per candidate
n : amount of candidates

As can be seen in figure 2.13 the cluster creation is the most time intensive part of the algorithm and is responsible for the largest part in the overall runtime. In contrast the runtime of the filters can almost be neglected.
When constraining the algorithm to only evaluate a maximum of five candidates per image and using formula 2.4 for that situation, it results in a worst-case runtime estimate of about 2 ms for the whole candidate generation. The average runtime can be estimated using 2.3 and an average amount of one candidate per image. This results in an average runtime estimate of 0.7 ms.

3 Classification Prerequisites

The following chapter is used to explain the basic terms needed to be able to evaluate a classifiers performance. It presents the software that was utilized for creating and evaluating the robot detection approaches assessed during this thesis and demonstrate which data was used and how it was processed for classification. 3.4 and 3.5 provide the theoretical basics on how NNs and CNNs work.

3.1 Basic Terms

In the field of machine learning there are a lot of different ways of visualizing the results of a classifier. One prominent way of presenting an overview of a classification outcome is the confusion matrix. Figure 3.1 shows such a confusion matrix for a binary classification problem with the two classes "yes" and "no".

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Figure 3.1: A confusion matrix for a two-class classification problem[8]

In order to stay in the context of this thesis, it can be assumed that the classifiers input data was robot candidate images. A "yes" label means that there was a robot in the image and "no" means there was no robot in the image. In this example the classifier was given 165 different candidate images and it generated an output label for each of them. Since the labels of the candidates were known upfront it is now possible to create the confusion matrix shown above.

By simply looking at each prediction and checking it's actual label it can be sorted into the correct matrix cell. If the classifier predicted "no" and it was actually "no" it goes in the top left cell. If the classifier predicted "yes" but it was actually "no" it goes into the top right cell, and so on. There are four basic terms which are important when comparing predicted and actual labels:

True Negative (TN): predicted "no" and it was "no"

False Negative (FN): predicted "no" but it was actually "yes"

True Positive (TP): predicted "yes" and it was "yes"

False Positive (FP): predicted "yes" but it was actually "no"

Adding these terms and the sums of the rows and columns to the matrix yields figure 3.2.

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Figure 3.2: Extended confusion matrix[8]

This matrix by itself is not suitable for comparing classification results. It does however provide the basis for various performance indicators.

Indicator	**Formula**
Error	$(FP + FN)/n$
Accuracy	$(TP + TN)/n = 1 - Error$
True Positive Rate	$(TP/FN + TP)$
False Positive Rate	$(FP/TN + FP)$
Precision	$(TP/FP + TP)$
Recall	$= True\ Positive\ Rate$
Sensitivity	$= True\ Positive\ Rate$
Specificity	$(TN/TN + FP) = 1 - False\ Positive\ Rate$

Table 1: Classification performance indicators[2]

Table 1 lists the most commonly used indicators. An explanation on which of these indicators are interesting in the course of this thesis is given in 4.2.

3.2 Used Software

This thesis can be split into two parts regarding the used software.

1. Testing and evaluating NNs
2. Implementation on the NAO

Most of the testing and research was done in Python using the *Caffe* machine learning library[5]. *Caffe* is a library developed and maintained by the *Artificial Intelligence Research Lab* of the University of Berkeley. It is mostly written in C++ but comes with a Python Application Programming Interface (API). The library allows for easy creation of complex NN structures and provides all the necessary functions for training and evaluation. It was used to automate the creation, training and testing of NNs for varying parameter sets. Another library which was used for several Python algorithms is *Scikit-learn*[15]. Scikit-learn provides very straightforward implementations of many different machine learning related algorithms. This library was mainly used for visualizations and data handling.

For running the created NNs on the NAO the C++ library *tiny-dnn* was used[25]. This library comes with a *Caffe* model converter which allows for a simple conversion of the models created with *Caffe*. Since this is a header-only library it is very lightweight and easy to implement on the NAO. Another feature this library provides is it's capability to use Streaming SIMD Extensions (SSE). An extended instruction set available for the NAO's processor which can greatly improve the runtime of the networks.

3.3 Data

As mentioned in 2.3 4302 candidate images were generated. These images are the basis for the data that is going to be used for training the networks. For testing 976 new candidate images were generated from data recorded in the HULKs laboratory.

For the purpose of labeling these candidates a Python script was written which shows one image at a time and depending on which button the user pushes, moves it to a folder named like the class it is supposed to belong to.

It is important to mention that only the information of the Y-channel of the YCbCr-images was used. The decision is based on [11], where the results showed that the color channels did not provide much useful information for classification. Using only the Y-channel also helps to keep the resources needed at a minimum because it reduces the used information by two thirds.

Another idea taken from [11] is to reduce the image width and height fed to the classifier. As shown there, the input images can be made much smaller and still produce good classification results. The image sizes analyzed during this thesis were created by using the median aspect ratio of all generated robot candidates, which was found to be 0.637. An example for this way of generating image could be: using an image width of 17 px the height becomes $17/0.637 \approx 27$ px.

A common way of handling the data necessary for training and testing classifiers is to split it into three parts.

Training set:
The set used for training the weights of a NN.

Validation set:
The set used for validating the model during training. This set is used to find at which state the model achieved the best results during training. It is also used to evaluate different model parameters.

Test set:
A set which is exclusively used to see how well the final chosen model generalizes. This set has to consist of data the classifier has never been in contact with before.

The data acquired was also split this way. 80% of the 4302 candidate images were used for the training set and the remaining 20% for the validation set. The way this was organized is explained in 4.1. The test set was formed from the 976 candidates generated from new data. This set always stayed the same and was never changed in any way. It consists of 49.6% background images and 50.4% robot images.

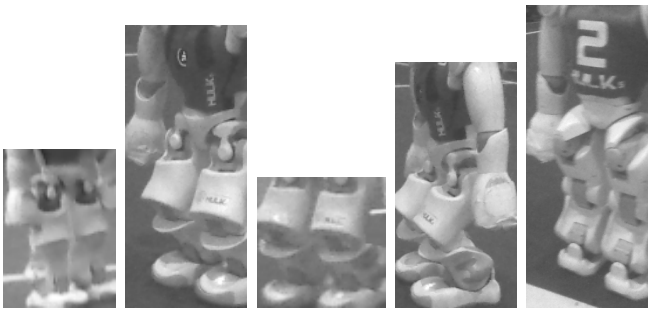


Figure 3.3: Robot class example images

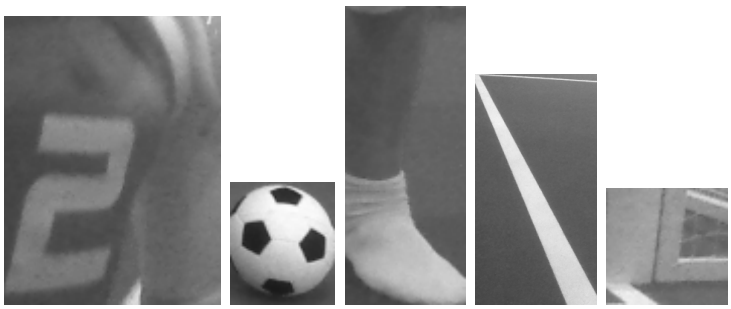


Figure 3.4: Background class example images

3.4 Neural Networks

Artificial neural networks are computational models inspired by the neurons in a human brain. Inside the brain the purpose of these neurons is to process information. By combining many of these neurons to a network, humans have the ability to make complex decisions based on the information the brain collects and processes. In the human brain each neuron has about 10^4 connections to other neurons all of them operating in parallel[2].

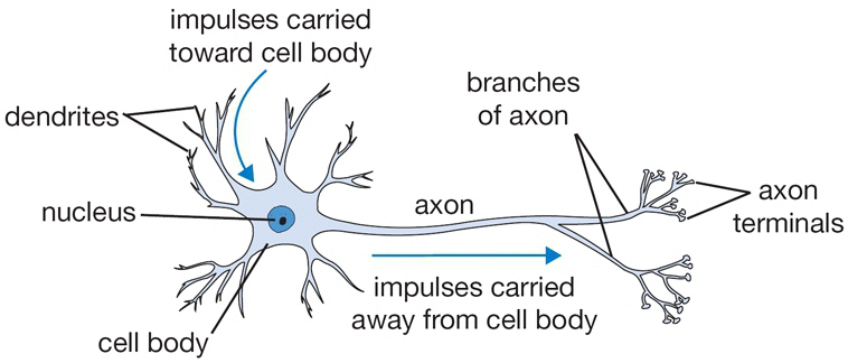


Figure 3.5: Sketched model of a biological neuron[10]

3.4.1 Neurons

Artificial NNs try to imitate these connections of the human brain. The smallest element in such a network is a single neuron. Figure 3.6 shows the mathematical abstraction of the biological model demonstrated in figure 3.5.

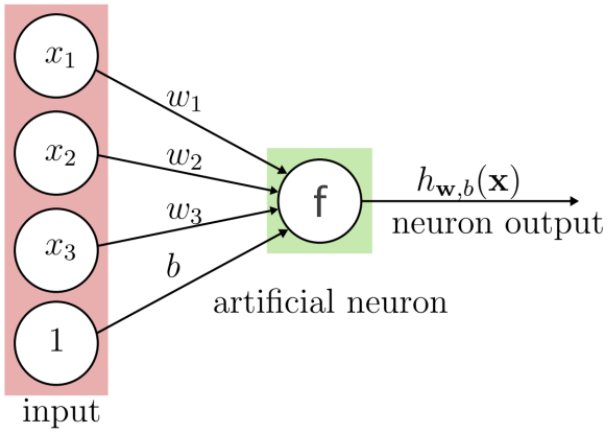


Figure 3.6: A single artificial neuron[12]

Here x_1, x_2 and x_3 represent the inputs and w_1, w_2 and w_3 the weights belonging to these inputs. By adding up the input-weight products and including the bias b as an additional

variable the neuron output $h_{w,b}$ is generated.

$$h_{w,b}(x) = f(\sum_{i=1}^{3} w_i x_i + b) \tag{3.1}$$

By defining $w_0 = b$ and creating input vector $x = [1, x_1, x_2, x_3]^T$ and weights vector $w = [w_0, w_1, w_2, w_3]$ the formula can be rewritten in a more compact form:

$$h_{w,b}(x) = f(w^T x + b) \tag{3.2}$$

3.4.2 Activation Functions

The function $f()$ in equations 3.1 and 3.2 symbolizes the activation function used by the considered neuron. The idea of the activation function is to introduce a non-linearity into the calculation which is necessary when approximating non-linear functions. There is a large variety of different activations functions but the most commonly used are the following:

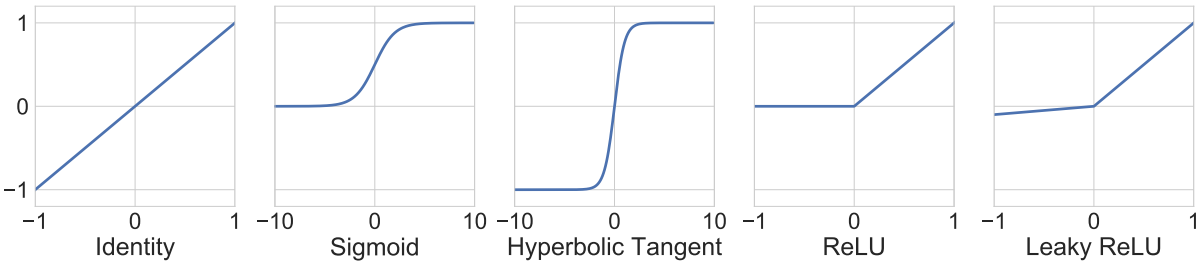


Figure 3.7: Overview over most common activation functions

Identity:

The identity function is the equivalent to not having any activation function at all. With this function the neuron's input equals it's output:

$$f(x) = x \tag{3.3}$$

Sigmoid:

The sigmoid function is a special case of the logistic function. It is a derivable relative of the step function which is bounded by the interval $[0, 1]$ and has a positive derivative at each point.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

Hyperbolic tangent:

Just like the sigmoid function the hyperbolic tangent is a S shaped function. The differences between the two are, that where the sigmoid function saturates inputs at 0 and 1 the hyperbolic tangent does this at -1 and 1. It is therefore a zero-centered function.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.5}$$

Rectified Linear Unit (ReLU):

The ReLU function has become one of the most popular activation functions for NNs. It computes:

$$f(x) = max(0, x) \tag{3.6}$$

which is the same as a thresholding at 0. This function is especially popular due to its simplicity and non-saturating character. It can however suffer from the "dying ReLU" problem. The "dying ReLU" problem describes the possibility of a weight-update during training in such a way that the neuron will never activate again. This can happen when setting the learning rate too high[10].

Leaky ReLU:

The Leaky ReLU is an attempt at avoiding the "dying ReLU" problem. It is basically a ReLU function with a minimal gradient for negative input values.

$$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases} \tag{3.7}$$

For this thesis α, was chosen to be 0.1 based on [12].

3.4.3 Networks

When connecting multiple neurons with each other in such a way that outputs of neurons become inputs of other neurons a neural network is formed. NNs are in most cases organized into layers of neurons as can be seen in figure 3.8.

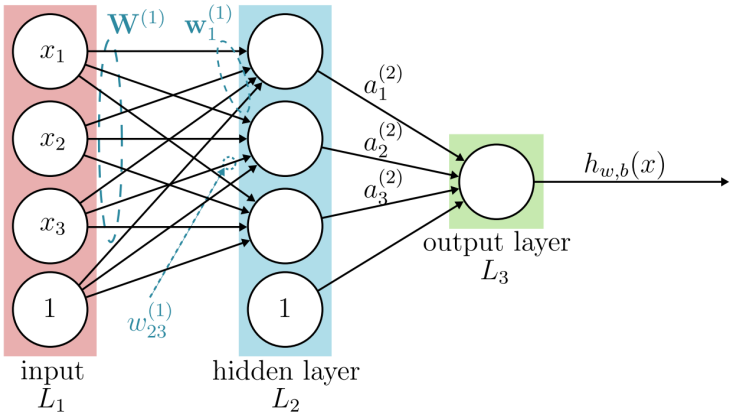


Figure 3.8: Model of a NN

The network shown above consists of three layers, an input-layer L_1, a hidden-layer L_2 and an output-layer L_3. It is important to know that the nodes in layer L_1 are no neurons since there is no computation happening in that layer. That is also the reason why such a network is referred to as a two-layer network. Another noticeable attribute of the network is that every node in one layer is connected to all nodes in the next layer. This attribute is why such a structure is called a fully-connected layer.

The notation for such a network also becomes more complex. A weight is now symbolized by $w_{ij}^{(l)}$, where l is the layer, j is the start-node and i is the end-node of the weighted connection. All weights that belong to a certain neuron j in layer $l+1$ can be summarized by the weight vector $w_j^{(l)}$. The same notation is also used for the biases. $b_i^{(l)}$ describes the bias connected to node i in layer $l+1$. The output of a neuron i in layer l is described by $a_i^{(l)}$.

Calculating the output of the network then becomes:

$$h_{w,b} = a_1^{(3)} = f(w_{11}^{(2)} a_1^{(2)} + w_{12}^{(2)} a_2^{(2)} + w_{13}^{(2)} a_3^{(2)} + b_1^{(2)}) \tag{3.8}$$

$$a_1^{(2)} = f(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)}) \tag{3.9}$$

$$a_2^{(2)} = f(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 + b_2^{(1)}) \tag{3.10}$$

$$a_3^{(2)} = f(w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 + b_3^{(1)}) \tag{3.11}$$

How such a network is trained is not demonstrated here because this thesis did not investigate this part. Information on training can be found in [2] [10], [12], and [20].

3.4.4 Influences on Runtime and Results

When analyzing NNs there are five main factors which have an influence on the classification outcome.

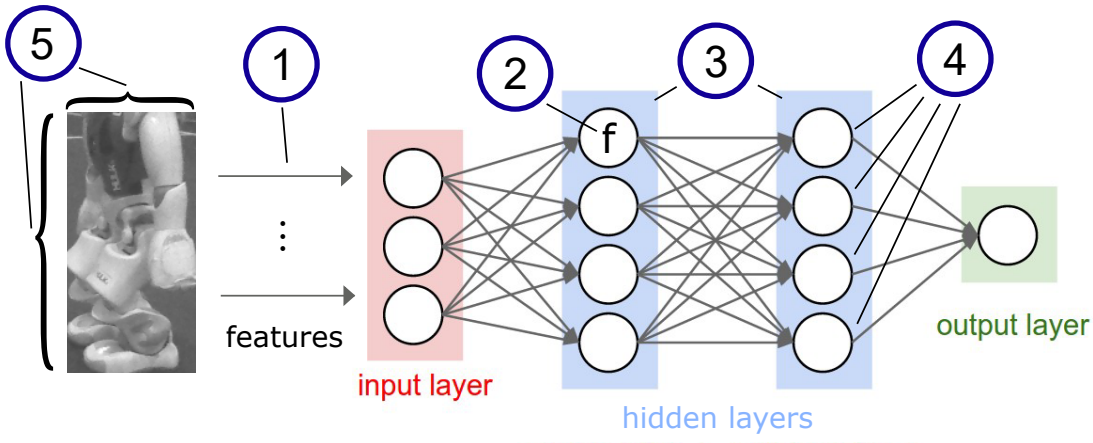


Figure 3.9: Visualization of influences on runtime and results of NNs. Based on [10]

1. The used features
2. The used activation functions
3. The amount of hidden layers
4. The amount of neurons in each layer
5. Input size (width and height of the used images)

The effects of each of these factors on the classification results and the necessary runtime will be examined in section 4.5.

3.5 Convolutional Neural Networks

CNNs have become very popular over the recent years. Especially due to Deep Learning Machines like Google's *AlphaGo* which recently won against a champion of the Chinese game *Go*[19].

CNNs are particularly famous for their capabilities in image recognition. Figure 3.10 shows the history of algorithms used for one of the worlds largest image recognition competitions, the ImageNet Large Scale Visual Recognition Competition (ILSVRC).

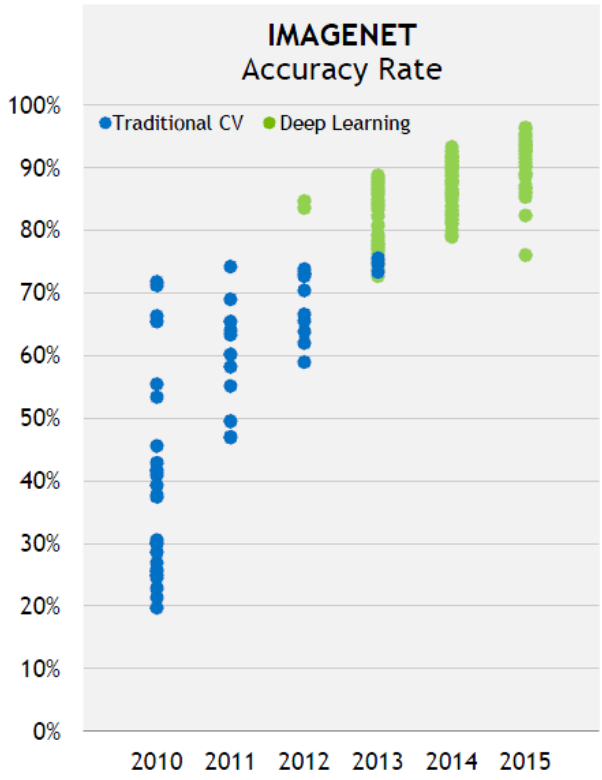


Figure 3.10: A history of the ImageNet accuracy rates[26]

CNNs were first used in 2012 for the ILSVRC and achieved accuracies much higher than the traditional algorithms. Only two years later all teams had already switched to using deep learning methods like CNNs.

As their names suggest, CNNs are a special kind of neural networks. They very much behave like conventional NNs. They are made up of neurons and have trainable weights and biases. Every neuron receives an input and produces an output using an activation function. They do however differ in many ways, the most significant being that they can learn the features necessary for recognizing objects in images.

CNNs make use of a variety of different layer types necessary for recognizing different patterns. Modern CNNs use up to 100 million parameters and usually consists of 10-20 layers, which is also the reason for the term *deep learning*[10].

The three different types of layers most CNNs consist of are *Convolutional Layers*, *Pooling*

Layers and *Fully-Connected Layers*. Fully-connected layers have already been explained in 3.4 and work for CNNs just like they were presented there.

3.5.1 Convolutional Layers

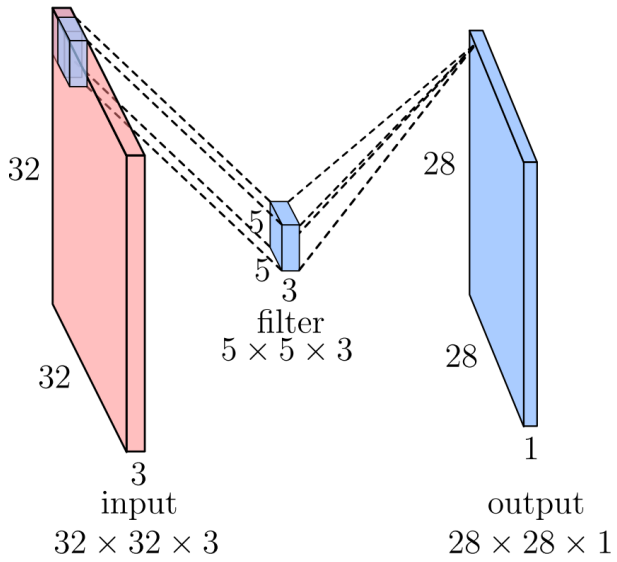


Figure 3.11: Sketch of a convolutional layer[12]

Figure 3.11 demonstrates the principle of a convolutional layer. The red block on the left is the network's input. For CNN-layers inputs are in most cases three dimensional volumes. The input here can be interpreted as a 32 by 32 pixels image with a depth of 3. The depth equals the channels of the image, e.g. red, green and blue for a RGB image. The smaller blue box in the middle resembles a convolutional kernel or filter. This is where the weights of the convolutional layer are located. A kernel is usually square and much smaller in width and height than the input. It does however always have the same depth as the input. Here the kernel is 5x5x3. This means that it holds 75 weights plus one bias which results in a total of 76 parameters. The kernel's weights and bias are trained like the weights in a conventional neural network. It's output is generated by sliding the kernel across the input and calculating a dot product of the weights with the inputs where it is currently located. The size of the output is determined by four factors: the input size h_{input}, the amount of kernels n, the size of the kernels h_{kernel} and the stride s. The stride describes how far the kernel moves with each iteration. A stride of 1 means that it moves one pixel per iteration whereas a stride of 5 results in a jump of five pixels per iteration. The output size assuming square kernels can be calculated via:

$$h_{output} = \frac{h_{input} - h_{kernel}}{s} \tag{3.12}$$

$$d_{output} = n \tag{3.13}$$

Using the formula it is easy to determine that the stride used for the example in figure 3.11 equals 1. As equation 3.13 shows, the depth of the output volume d_{output} is only determined by the amount of kernels n. So each kernel creates one output slice. When using multiple kernels the output slices are stacked and form a volume.

The kernel's output slices are often referred to as feature maps. Each of the kernels learns to extract a specific feature in the image. This could be things like vertical edges, horizontal edges or colors. By using the output of a convolutional layer as the input for another layer the network gains the ability to combine features and thus is able to find specific characteristics of objects contained in the images.

3.5.2 Pooling Layers

Pooling layers are often used in-between successive convolutional layers. Their objective is to reduce the size of the data that is being processed to lower the amount of parameters needed and reduce the overall computation cost.

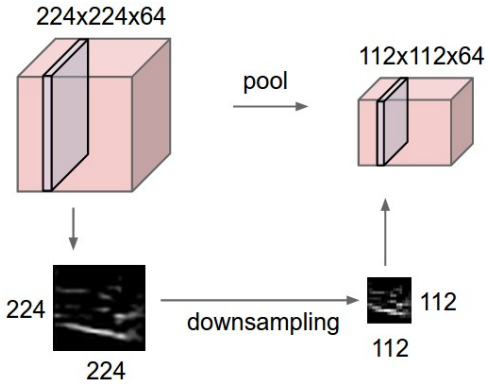


Figure 3.12: Demonstration of the downsampling effect of the pooling layer[10]

Figure 3.12 demonstrates the downsampling effect achieved by such a layer. Not unlike the convolutional layers the pooling layers use a kernel that slides over the input. It does not however have weights and calculate a dot product but it rather applies one single operation on the area it is looking at. There are a couple of different operations which can be applied like averaging or applying a L2-Norm, but the most popular is max pooling.

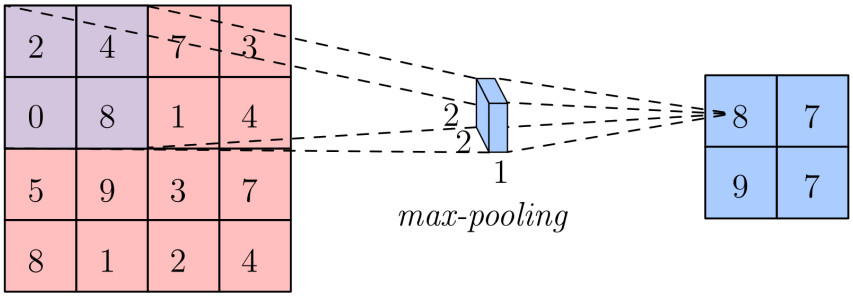


Figure 3.13: Sketch of a max pooling layer[12]

Max pooling chooses the highest value of all values it is regarding and uses that as it's output. Figure 3.13 shows how max pooling works. The most common way of applying max-pooling is with a pooling window of size 2 and a stride of 2. This makes sure that the sliding window never looks at overlapping parts. Applying the layer this way cuts the data by 75% as can be seen in figure 3.12 where the data was reduced from $224 \cdot 224 \cdot 64 = 3211264$ to $112 \cdot 112 \cdot 64 = 802816$ data points.

Combining convolutional, pooling and fully-connected layers is the basis of all CNNs. An example of such a network can be viewed in figure 3.14. The CNN shown here is called AlexNet, this is the network that won the ILSVRC in 2012.

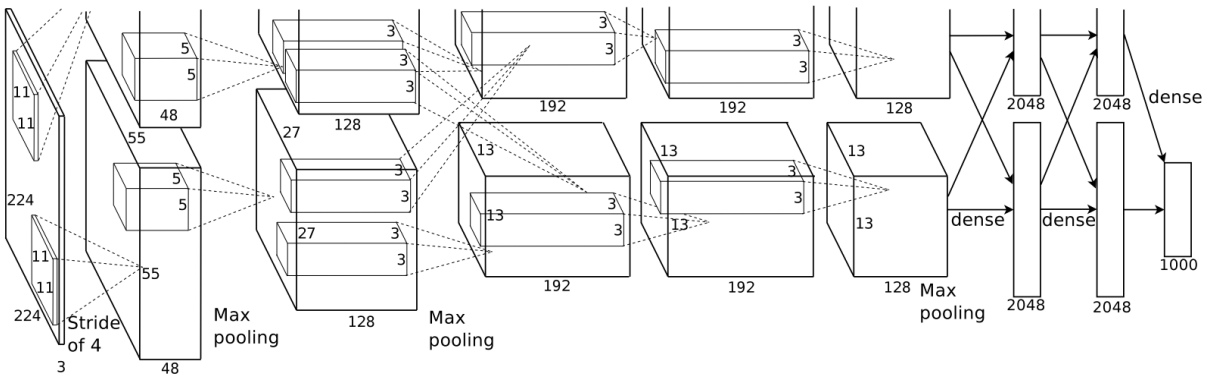


Figure 3.14: Structure of AlexNet[9]

3.5.3 Influences on Runtime and Results

When analyzing CNNs there are seven main factors which have an influence on the classification outcome.

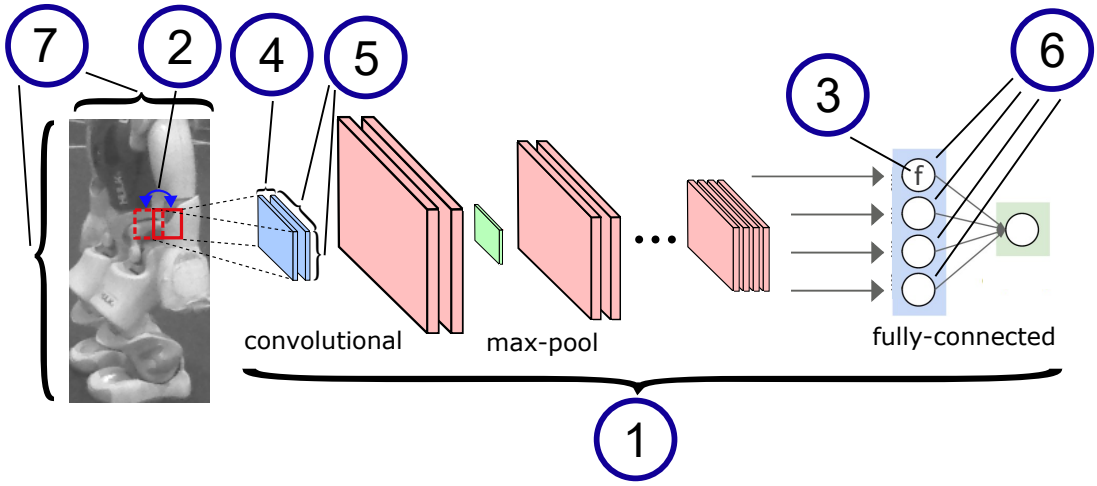


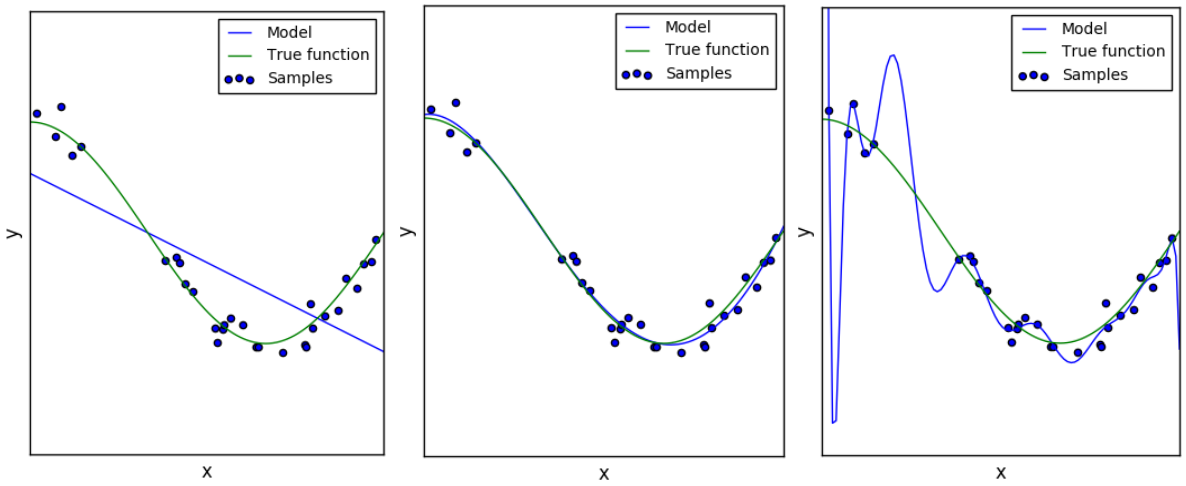
Figure 3.15: Visualization of influences on runtime and results of CNNs

1. The used network structure
2. The stride
3. The used activation functions
4. The amount of convolutional kernels
5. The size of the convolutional kernels
6. The amount of hidden neurons in fully-connected layers
7. Input size (width and height of the used images)

The effects of each of these factors on the classification results and the necessary runtime will be examined in section 4.6.

3.6 Overfitting

Overfitting is a problem discovered commonly when using machine learning. It describes the problem that a given model can fit it's training data so perfectly, that it looses the ability to generalize. Figure 3.16 shows this effect schematically.



(a) Underfitting (b) Good fit (c) Overfitting

Figure 3.16: Plot showing ways of fitting polynomial data samples. (a): Using a linear approximation which underfits the true function.(b): A polynomial of degree 4 fits the true function almost perfectly. (c): A polynomial of degree 15 overfits the data.

When using a neural network with possibly tens of thousand of degrees of freedom it is easy to overfit, especially when using only small amounts of training data. The effect this can have on the classification result is shown in figure 3.17.

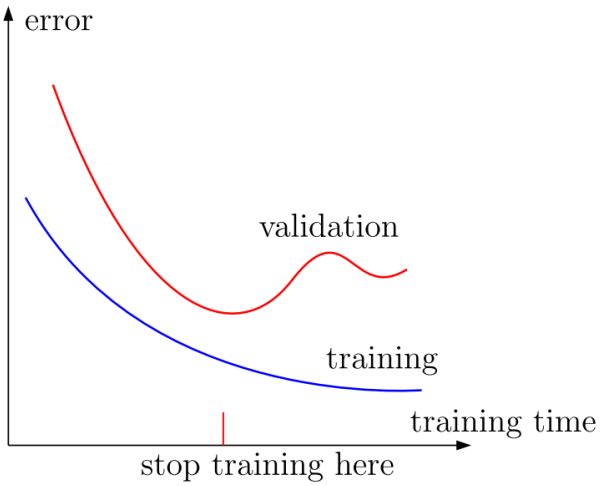


Figure 3.17: Effects of overfitting on the classification outcome of a classifier [12]

Figure 3.17 shows that when comparing the training error with the error on the validation set it can happen that at a certain point in training the error on the validation set begins to get worse. This is what happens when the classifier overfits the training data. Since it has never seen the validation data before, the increase in error on the validation set while the training error keeps getting smaller clearly shows that the classifier looses it's ability to generalize.

When not addressed, overfitting is a problem that can have a great impact on how well a classification achieves. The methods that were used to make sure this doesn't happen are explained in 4.3.

4 Tests and Evaluation

The following chapter describes how the different classification approaches were tested and presents the collected results. First it will be explained how the general procedure of training and testing a network was handled. It will also demonstrate how the quality of each classifier was determined by introducing a scoring function. In 4.3 there will be a brief explanation on what measures were taken to prevent the problem of overfitting. The chapter is concluded by the evaluation of all influences on classification results and runtime for feature-based classification in 4.5 and for CNNs in 4.6.

4.1 Execution

This section is going to give a short overview about how the testing was executed. It is divided in the two phases each network went through: "Training and Validation" and "Testing".

4.1.1 Training and Validation

For training and validating the networks, a k-fold-cross validation with $k = 5$ was executed. A k-fold cross-validation describes the process of splitting the given data in such a way that it consists of k different training - validation splits. It is split such that each data point is part of the training set $k - 1$ times and part of the validation set once [20].

For creating and training different networks a script was written which automates this process. The parameters that can be passed to the script are:

- Input image size
- Activation function
- Amount of hidden neurons
- Stride (CNNs only)
- Kernel size (CNNs only)
- Amount of kernels (CNNs only)

All of these parameters can be passed as lists such that the script will create and train a network for each combination possible. The training was carried out using mini-batches of size 200 which means that for each training iteration the network is fed 200 random images taken from the training set on which it computes it's weight updates. The training of a network stops as soon as 3000 iterations are done.

During the training process a snapshot of the current model is taken after every 100 iterations. A snapshot saves all current weights of the network and the current solver

state. In total a training run which involves 3000 iterations saves 30 snapshots along the way.

Validation is done by taking all snapshots created during training and evaluating them on the validation set. From all evaluated snapshots the script saves the snapshot which achieved the highest accuracy in a separate file and labels it as the best iteration for this network.

After training and validating five times the results of the five best snapshots are averaged and written to a log file for later evaluation. The network runtime is added to this log via another script that measures the time each network needs on the NAO.

The validation script also creates different plots for visual evaluation. 4.1 shows a comparison plot between training and validation accuracy. An example of the visualized weights of a convolutional layer is demonstrated in A.2.

Figure 4.1: Training and validation accuracy development during training of a CNN

4.1.2 Testing

For testing, chosen networks are retrained on the complete training and validation data and then evaluated on the test set. The retraining is done to make use of the full data that was previously split in a training and validation set.

4.2 Evaluation Criteria

The classification performance of the different NNs will be evaluated with regard to three main factors:

1. False Positive Rate
2. Runtime
3. Recall (True Positive Rate)

FPR is inarguably the most important criterion for measuring the performance of a robot detection. If a robot was to detect other robots in regions where there are none it is going to be making bad decisions and it might be better to not be able to detect any robots at all. One example of such a bad decision would be when a robot is standing right in front of the goal without any other robots around and instead of it shooting at the goal it decides to move away because of a non existent robot it detected in front of itself.

The second most important criterion is the overall forward computation time a network needs on the NAO to evaluate a single robot candidate. Due to having to evaluate about 30 frames per second, the time that can be spent on evaluating one single incoming image is very limited. Robot detection definitely is not a top priority function compared to other modules like generating motion trajectories, detecting the ball, the goal or field lines. Therefore the time the robot detection needs should be kept at a minimum.

The third and last criterion that is going to be used for evaluation is the True Positive Rate (TPR), also called Recall. After making sure that false positives and the runtime are at a minimum the algorithm still needs to be able to correctly detect a robot as a robot. Even though this is the least important factor it is still very desirable to maximize it.

In order to be able to tell which networks are good enough to be considered for the robot detection, several constraints were set each network had to fulfill.

For a network to be considered at all, the following limits were set:

- Maximum FPR: $FPR_{max} = 10\%$
- Maximum runtime: $t_{max} = 1.5$ ms per candidate
- Minimum recall: $R_{min} = 70\%$

For a network to be considered a very good choice, it has to fulfill:

- Maximum FPR: $FPR_{max} = 5\%$
- Maximum runtime: $t_{max} = 0.75$ ms per candidate
- Minimum recall: $R_{min} = 85\%$

Assessing the quality of a network is done by using a rating function which calculates a score for each network. Using the minimum requirements for a valid network $FPR_{max} = 0.1$, $t_{max} = 1.5$ms and $R_{min} = 0.7$, the score is calculated as follows:

$$Score(net) = \begin{cases} (w_{fpr} \cdot FPR_{score} + w_t \cdot t_{score} + w_r \cdot R_{score}) \cdot 100 & C \neq 0 \\ 0 & C = 0 \end{cases} \tag{4.1}$$

$$FPR_{score} = max\left(0, \left(1 - \frac{FPR}{FPR_{max}}\right)\right) \tag{4.2}$$

$$t_{score} = max\left(0, \left(1 - \frac{t}{t_{max}}\right)\right) \tag{4.3}$$

$$R_{score} = max\left(0, \left(1 - \frac{1 - R}{1 - R_{min}}\right)\right) \tag{4.4}$$

$$C = FPR_{score} \cdot t_{score} \cdot R_{score} \tag{4.5}$$

net: the NN that is being evaluated
w_{fpr}: weight for the FPR-score
w_t: weight for the runtime-score
w_r: weight for the recall-score
FPR: FPR achieved by net
t: runtime achieved by net
R: recall achieved by net
C: a control variable to check whether any score equals 0

By using the C variable the function makes sure that as soon as a network does not achieve the minimum requirements the score equals 0. If the network does fulfill the minimum requirements, the score will be a number between 0 and 100, 100 being the best score obtainable. By first calculating a score for each criterion the function determines how well a network achieves within the bounds of that criterion. Using the weights w_{fpr}, w_t and w_r allows to adjust how much influence each criterion has on the final score. For the course of this thesis the weights were chosen as:

- $w_{fpr} = 0.6$
- $w_t = 0.3$
- $w_r = 0.1$

This is a choice that was made to represent the importance of the criteria mentioned before. If in hindsight, it is decided that the runtime was more important than the FPR after all, these weights can be set accordingly and the networks can be reevaluated.

4.3 Measures against Overfitting

As described in 3.6 overfitting is a common problem that can have negative effects on the results of a classifier. In order to prevent this from happening there are several methods that can be implemented.

One of these methods was already hinted at in figure 3.17. At the dip of the validation error the figure states “stop training here”. This describes one common way of preventing overfitting. By simply stopping the training when the validation error stops improving the network can be kept in a state where the overfitting did not have an effect on the networks generalization capabilities yet. This method was also used for this thesis. As described in 4.1.1 the script that was used to validate the training of the different networks checks every snapshot of the network that was saved during training. By then choosing the iteration where the network achieved the best results on the validation set, it is made sure that overfitting is kept at a minimum. Looking at figure 4.1 it is easy to see that for this example the validation accuracy reaches it’s peak at about 1000 iterations. Afterwards it drops slightly and then stagnates for the rest of the training. For this case the said validation script picked the model saved at iteration 1000 and marked this for further use.

Making use of a five-fold cross validation also makes sure that the training does not overfit to a certain data split. Since every data point is part of the training set four times and part of the validation set once the results are not biased towards a single split.

By using a test set for a final evaluation which consists only of data the classifier has not seen before and which was collected under different circumstances than the training and validation data, the chance of overfitting is further decreased.

There are also other methods of avoiding overfitting like regularization and dropout, both of which could further improve the classification results. However, due to these methods being complex an investigation did not fit into the scope of this thesis.

4.4 Feature Calculation

In order to be able to determine whether a candidate image contains a robot or not a classifier needs to learn how to differ between robot and non-robot images. The way this is achieved is by extracting features from the images that describe the characteristics of robot and non-robot images which are then used as an input for the classifier. One such feature could for example be the most common Y-value in an image. Since the NAO robots are mostly white and gray, an image holding a NAO will consist of a different spectrum of Y-values than a non robot image. Due to the resource constraints on the NAO the final selection of features should only contain such features that differentiate well between robot and non-robot images and require low computation costs.

Finding the appropriate features was done in three steps:

1. Extracting a pool of features to choose from
2. Determine which features provide the best classification results
3. Select a set of features and test the overall runtime

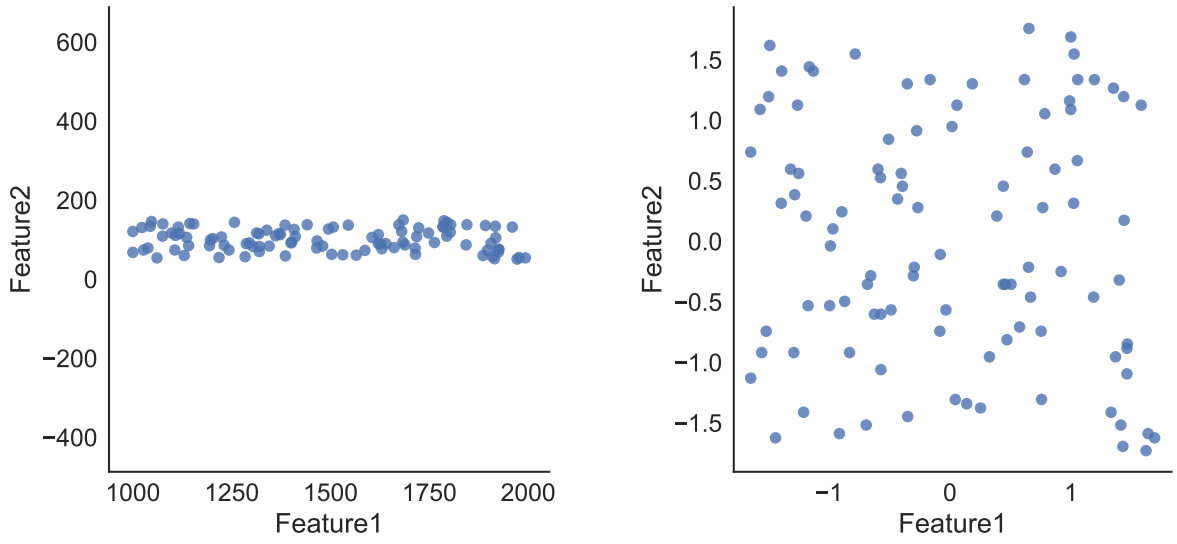
4.4.1 Standardization

A standardization algorithm is used for insuring that no feature dominates the others by having a higher range of values. The way this is done is by calculating the mean μ and the standard deviation σ for each feature in the training set[7]. Applying the formula

$$X_{std} = \frac{X - \mu}{\sigma} \tag{4.6}$$

on each feature X, the features are scaled to zero-mean and a standard deviation of one. Figure 4.2 demonstrates an exemplary application of the standardization.

The mean and the standard deviation of each feature in the training set has to be remembered so it can be applied for scaling the features in the validation and training set in the same way. This is done by writing the values to a JSON file which can then be read and applied when necessary. A sample file can be viewed in A.3.



(a) Unscaled features

(b) Scaled features

Figure 4.2: Two features before and after standardization. In (a) feature 1 dominates feature (2) due to it's much larger value range.

4.4.2 Feature Extraction

A large amount of features was extracted so the selection algorithms have many features to choose from. First the following intermediate features were calculated:

- Histogram of Y-values
- Gradients in y-direction
- Gradients in x-direction
- Histogram of gradients in y-direction
- Histogram of gradients in x-direction
- Ten most common Y-values
- Ten most common gradients in y-direction
- Ten most common gradients in x-direction
- Magnitude of the Fourier-Transform
- Phase of the Fourier-Transform

Then variance and entropy were calculated for each of the above. The formula for variance is given in 2.1. Entropy was calculated as follows:

Let F be the set of values of an intermediate feature.

$$H(F) = -\sum_{i=1}^{n} p(x_i) \cdot log_2(p(x_i)) \tag{4.7}$$

$$p(x_i) = \frac{a(x_i)}{n} \tag{4.8}$$

$H(F)$: entropy of a set F
x_i: a single value from F
$p(x_i)$: the probability of occurrence of x_i
n: amount of values in F
$a(x_i)$: occurrences of x_i in F

The following features were calculated as an addition to the 20 features resulting from the above:

- Mean of all Y-values in image
- Mean of the ten most common Y-values
- Mean of the ten most common gradients in y-direction
- Mean of the ten most common gradients in x-direction
- Spectral entropy

Where the formula used for calculating the mean is:

$$\mu(F) = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4.9}$$

$\mu(F)$: mean of a set F

Spectral entropy is a feature based on the Fourier-Transform of the image. It is calculated in three steps[28]:

1. Calculate the Power Spectral Density (PSD) $\hat{P}$ for all Y-values y_i in the Y-channel of an image using their Fourier-Transform $X(y_i)$:

$$\hat{P}(y_i) = \frac{1}{N} \cdot |X(y_i)|^2 \tag{4.10}$$

N: amount of Y-values in image

2. Obtain the power spectral density distribution p_i for each y_i by normalizing $\hat{P}(y_i)$:

$$p_i = \frac{\hat{P}(y_i)}{\sum_{i=1}^{N} \hat{P}(y_i)} \tag{4.11}$$

3. Using the formula for entropy 4.8 on the result $P = [p_1, p_2, ..., p_N]$ yields the spectral entropy $S(P)$:

$$S(P) = -\sum_{i=1}^{n} p_i \cdot log_2(p_i) \tag{4.12}$$

Combining all features results in a total of 25 features which need to be evaluated.

4.4.3 Feature Selection

For evaluating the features two programs were written which automatically choose those features that have a positive impact on the classification. A NN with 10 hidden neurons and ReLU activation function was used as the classifier for the feature selection.

The algorithms evaluate the used features by calculating a score from the achieved FPR, recall and accuracy. The calculated score is bound to the interval $[0, 1]$, 0 being the worst score and 1 the best.

$$F_{score} = 0.5 \cdot (1 - FPR) + 0.3 \cdot Recall + 0.2 \cdot Accuracy \tag{4.13}$$

As is obvious from equation 4.13 the FPR is weighed the highest, followed by recall and accuracy. This is done to account for the importance of a very low FPR for a successful robot detection. For every evaluation the NN was trained and tested ten times. The score was then calculated on the averaged results.

The first algorithm applies a feature reduction. It does this by starting with a NN trained on the full set of features. It then calculates the score for the classification result and retrains the classifier with one feature less. If the resulting score does not drop by more than 0.01, the feature is marked for removal and the program proceeds to check the next feature. If the score does however drop by more than 0.01 the feature is marked as necessary before moving on.

Algorithm 1 Feature reduction

Input:
$features$ ▷ A list of all features
$classifier$ ▷ The used classifier

```
1: result ← classifier.train_and_test_10_times(features)
2: reference_score ← evaluate(result)
3:
4: for each feature in features do
5:     features.remove(feature)
6:     result ← classifier.train_and_test_10_times(features)
7:     score ← evaluate(result)
8:     if score < reference_score − 0.01 then
9:         feature.significant ← False
10:    else
11:        feature.significant ← True
12:    end if
13:    features.add(feature)
14: end for
```

After the program finishes a log is written which contains all the features flagged as significant.

The second algorithm works the other way around. It starts training the classifier with only one of the 25 features at a time. The feature which scores the highest is then remembered and will be kept for the second training round. In the second round it adds one of the remaining features to the one that scored best in the first round and again checks which feature achieves the highest score. The procedure is repeated until either all features were checked or no feature improves the score any further.

Algorithm 2 Feature picking

Input:

features ▷ A list of all features
used_features ▷ An empty list
classifier ▷ The used classifier

```
1:  done ← False
2:
3:  while done ≠ True do
4:      max_score ← 0
5:      best_feature ← 0
6:
7:      for each feature in features do
8:          used_features.add(feature)
9:          result ← classifier.train_and_test_10_times(used_features)
10:         score ← evaluate(result)
11:
12:         if score > max_score then
13:             max_score ← score
14:             best_feature ← feature
15:         end if
16:
17:         used_features.remove(feature)
18:     end for
19:
20:     if best_feature ≠ 0 then
21:         used_features.add(best_feature)
22:         features.remove(best_feature)
23:
24:         if features.empty() then
25:             done ← True
26:         end if
27:     else
28:         done ← True
29:     end if
30: end while
31: return used_features
```

Both algorithms were run 10 times. Evaluating the written logs showed that reduction algorithm chose an average of 6.7 features with a minimum of 5 and a maximum of 14. The picking algorithm picked 11.8 features at average with a minimum of 5 and a maximum of 18. The varying amounts of features picked can be explained by the fluctuations in the averaged classification results. Since training a NN is non-deterministic the outcome always varies slightly which is why it is important to run the algorithms multiple times. The higher average for the picking algorithm can be explained by it accepting another feature even if it only slightly improves the score, whereas the reduction algorithm uses a fixed threshold.

Table 2 lists how often each feature was picked by each algorithm and overall. Features which were not chosen at all are not listed.

Feature	**Picking**	**Reduction**	**Overall**
Variance histogram of gradients x	7	9	**16**
Variance histogram of gradients y	10	6	**16**
Spectral entropy	5	10	**15**
Variance 10 most common Y-values	10	4	**14**
Variance gradients y	6	4	**10**
Variance gradients x	6	4	**10**
Mean 10 most common gradients y	9	0	**9**
Entropy gradients y	3	4	**7**
Entropy fourier magnitude	6	0	**6**
Entropy histogram of Y-values	5	0	**5**
Variance fourier magnitude	5	0	**5**
Variance histogram of Y-values	3	1	**4**
Entropy gradients x	1	3	**4**
Entropy 10 most common Y-values	4	0	**4**
Variance 10 most common gradients y	4	0	**4**
Entropy histogram of gradients x	4	0	**4**
Entropy 10 most common gradients y	3	0	**3**
Variance 10 most common gradients x	1	1	**2**
Entropy histogram of gradients y	2	0	**2**
Mean 10 most common Y-values	2	0	**2**
Mean of Y-channel	1	0	**1**

Table 2: Feature selection results

Based on these results three different combinations of features were chosen and tested.

Combination 1:

1. Variance histogram of gradients x
2. Variance histogram of gradients y
3. Variance gradients x
4. Variance gradients y

Combination 2:

All features from Combination 1 with spectral entropy as a fifth feature.

Combination 3:

All features from Combination 1 with variance of the 10 most common Y-values as a fifth feature.

The reason for choosing the combinations like this is that the two features which gave the best overall results, the variance of the histograms of gradients in x and y direction, depend on calculating the gradients in x and y direction first. Calculating the variance of those gradients only adds very little computing time which is why Combination 1 consists of those four features. Combination 2 and 3 add the features which ranked third and fourth in the evaluation. Those two features need extra calculations which is why it is especially interesting to see how the classification results change when using one or the other or even none of the two.

	Combination 1	Combination 2	Combination 3	All features
False Positive Rate	11.07%	8.3%	11.05%	8.2%
Recall	88.4%	88.5%	90.8%	89.2%
Accuracy	89.3%	90.9%	90.3%	91.2%
Score	0.8885	0.9058	0.8978	0.909

Table 3: Results of testing different feature combinations

Table 3 shows the results of the different combinations. In addition the last column also lists the results when all 25 features are being used. Again, the shown outcomes are the averages of training and testing ten times on varying train-validation splits.

As can be seen the best score is achieved when using all 25 features. Combination 1 does however score comparably well while using only four features. This demonstrates that there is a lot of redundancy in the information provided by the 25 features. The score of Combination 2 shows that using spectral entropy results in about 3% less FPR and a score which gets very close to using all 25 features. The feature added for Combination 3 does not noticeably affect the FPR but rather increases recall and accuracy slightly. Since achieving a low FPR is the most important criteria for a successful robot detection, Combination 2 was selected for further testing.

4.5 Feature-based Classification using Neural Networks

In order to evaluate the general influence the parameters outlined in 3.4.4 have on classification results and runtime, the following sections will show the results of all networks tested for the feature-based classification. First it will present the cross-validation results which will be used to determine which combination of network parameters promise the best results. This will be followed by evaluating the five best parameter combinations on the test set.

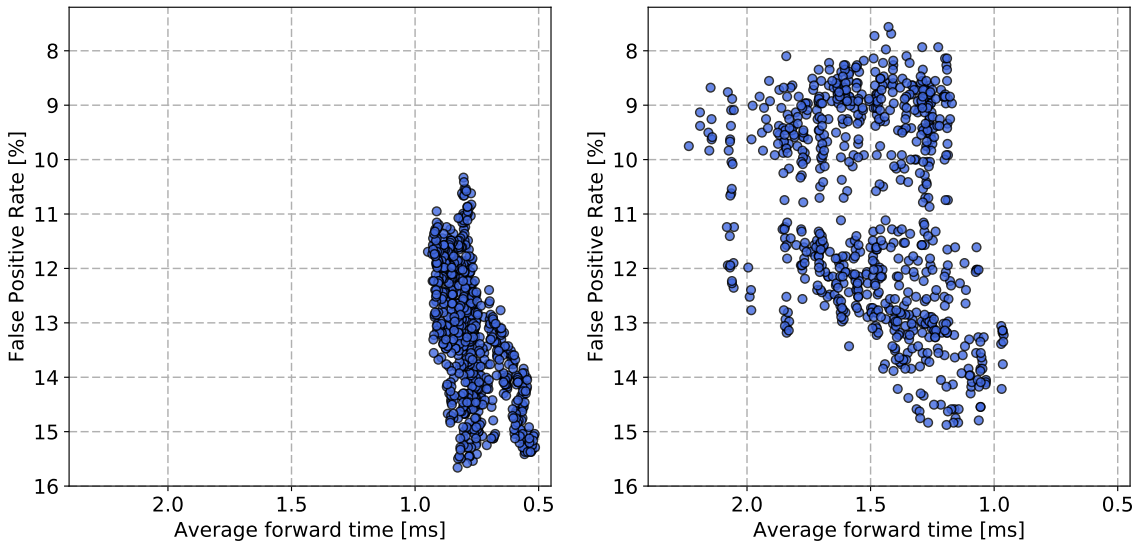
The following sets of parameters were tested:

- Activation functions = [Identity, Sigmoid, TanH, ReLU, Leaky ReLU]
- Amount of hidden neurons = [1, 2, 4, 6, 8, 10, 20, 50, 100]
- Input image sizes = [9x14, 10x16, 11x17, 12x19, 13x20, 14x22, 15x24, 16x25, 17x27, 18x28, 19x30, 20x31, 21x33, 22x35, 23x36, 24x38, 25x39]

For each possible combination of these parameters a network was trained. This results in a total of 765 different NNs.

4.5.1 Evaluation of used Features

Features were already evaluated in table 3 with regard to their FPR and recall but so far no statement regarding the runtime was made. In figure 4.3 each blue dot represents a single NN's cross-validation results. As can be seen the plots reflect the results from table 3. Using the spectral entropy feature in combination 2 does allow for some networks to achieve a FPR of about 3% better than without that feature. For reasons of comparability with CNNs the forward time used for the plots includes feature calculation as well. It is easy to see that the trade off that is made for a higher FPR is a much higher computation time resulting from calculating the Fast Fourier Transform (FFT) needed for the spectral entropy feature. Networks using this feature need between 0.9 and 2.25ms whereas networks without the feature only need 0.5 to 0.95ms.



(a) Combination 1

(b) Combination 2

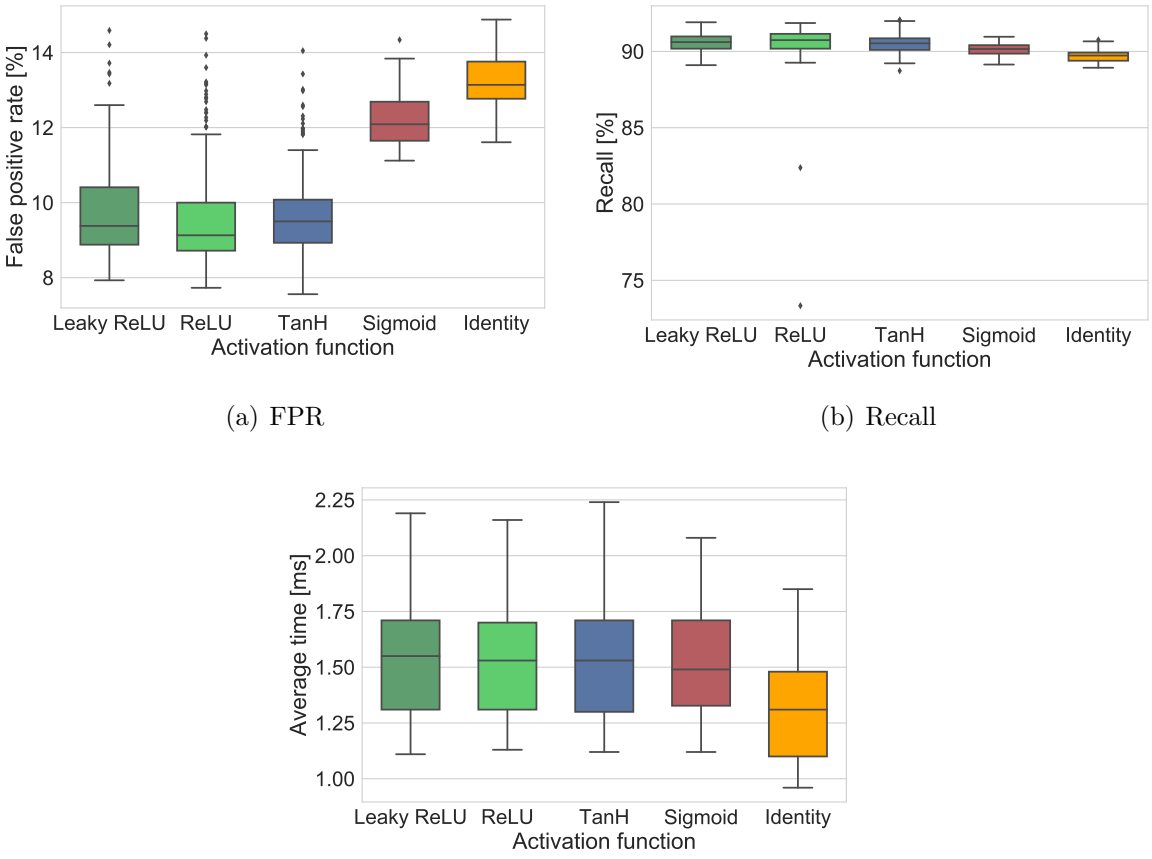
Figure 4.3: Cross-validation results for feature combination 1 and 2

Since no network using combination 1 was able to achieve a FPR under 10% which was set as a minimum requirement in 4.2, only combination 2 will be taken into account for further evaluation. The complete results with regard to all three evaluation criteria for both combinations can be viewed at A.4 and A.5.

4.5.2 Evaluation of Activation Functions

According to [10] ReLU is the best choice for an activation function, Sigmoid should never be used and TanH should be expected to score worse than ReLU. In order to find out whether this recommendation is true the networks were tested using the different activation functions explained in 3.7.

The results that were achieved can be viewed in figure 4.4.



(a) FPR

(b) Recall

(c) Computation times

Figure 4.4: Results of all 765 tested NNs with regard to the used activation functions

In terms of FPR the ReLU, leaky ReLU and TanH functions scored similar whereas sigmoid and identity scored much worse.

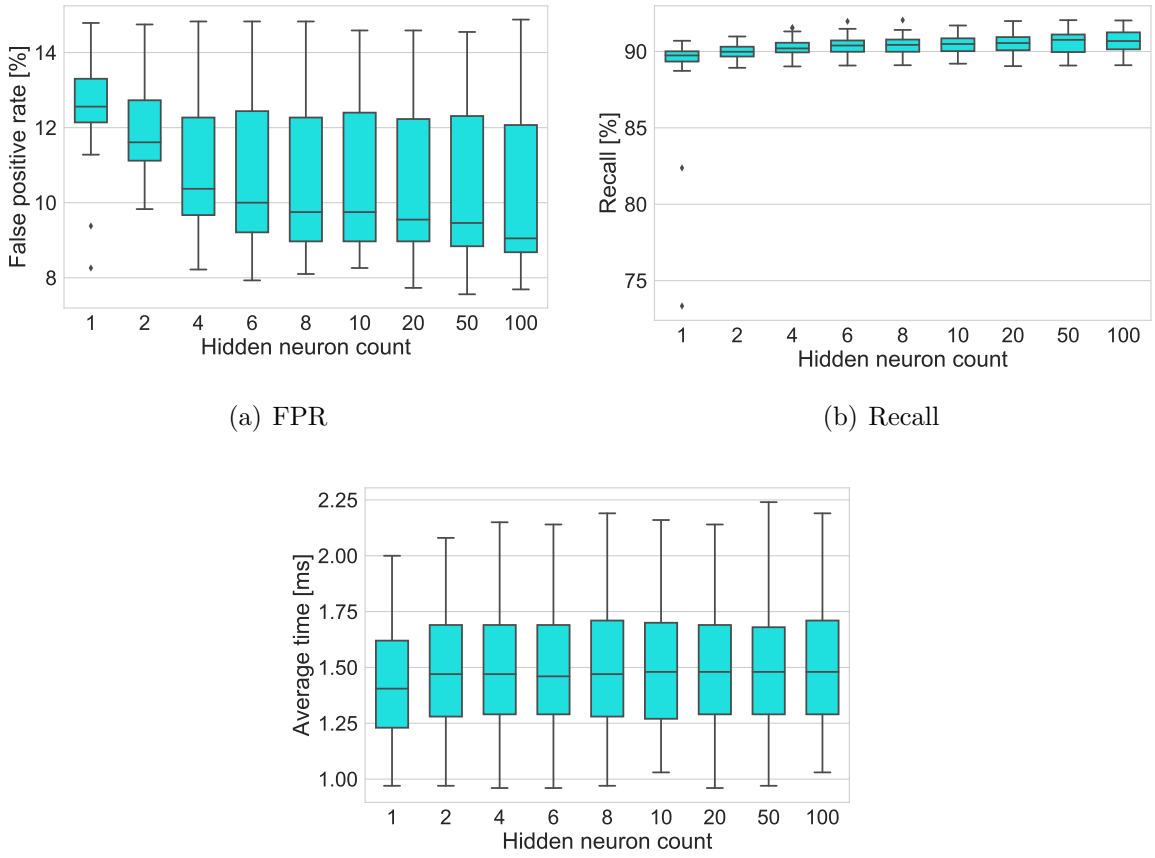
The same effect can be seen in plot (b). Again the ReLU, leaky ReLU and TanH functions all achieved comparable recall rates while sigmoid and identity functions produced lower scores. All five did however manage to obtain very high recall rates.

Looking at the average runtime the identity function proved to be the fastest. All other four functions only showed neglectable differences.

As recommended by [10] the ReLU function proved to be one of the best functions. It did however not turn out to be much better than the leaky ReLU or TanH functions.

In general any of these three functions can be chosen and will be able to produce good results.

4.5.3 Evaluation of Neuron and Hidden Layer Count



(a) FPR

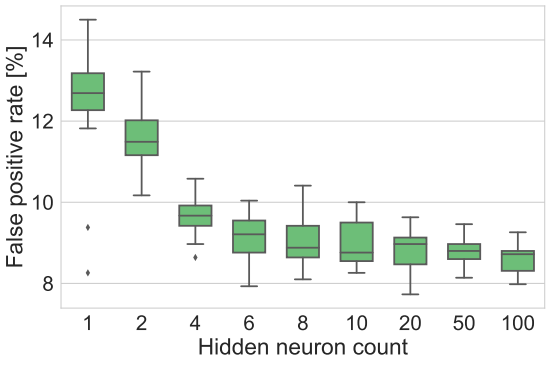
(b) Recall

(c) Computation times

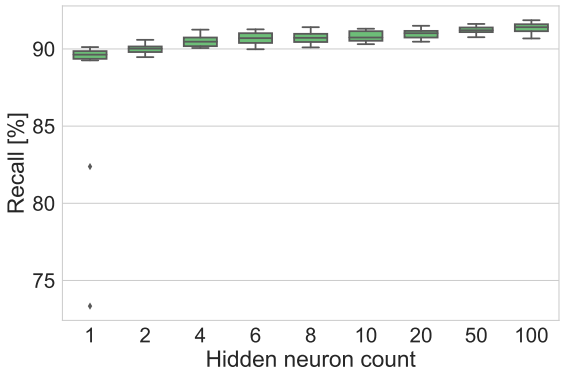
Figure 4.5: Results of all 765 tested NNs with regard to the used neuron count

Figure 4.5 demonstrates the effect of using different amounts of neurons in the hidden layer. While it is possible to see that increasing the amount of neurons leads to a lower FPR and slightly higher recall, the effect is better visible when looking at only those networks which use ReLU activation as shown in figure 4.6. The difference between the plots is caused by the networks using identity and sigmoid activation. These networks cannot make use of the higher amount of neurons as can be seen in A.11 and A.12.

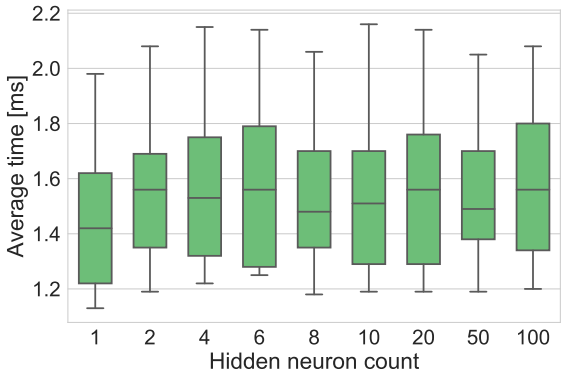
Figure 4.6 shows that the higher the amount of neurons in the hidden layer the better the results. While recall only improves slightly with rising neuron count the FPR is reduced significantly. It is however also visible that the reduction in FPR saturates in such a way that increasing beyond 100 neurons would most likely not reduce the FPR much further.



(a) FPR



(b) Recall



(c) Computation times

Figure 4.6: Results with regard to the used neuron count for NNs using ReLU activation

Regarding the runtime, increasing the amount of neurons does not appear to increase the computation time. The results for each neuron count vary approximatively between 1.2 and 2.1 ms. As the next section will show, the runtime is much more dependent on the input size than the neuron count.

Since a high neuron count proves to increase recall and decrease FPR, while having almost no effect on the runtime, it is safe to say that using a high amount of neurons is a good choice.

Due to the already very high computation times the addition of a second hidden layer was not investigated any further.

4.5.4 Evaluation of Input Dimensions

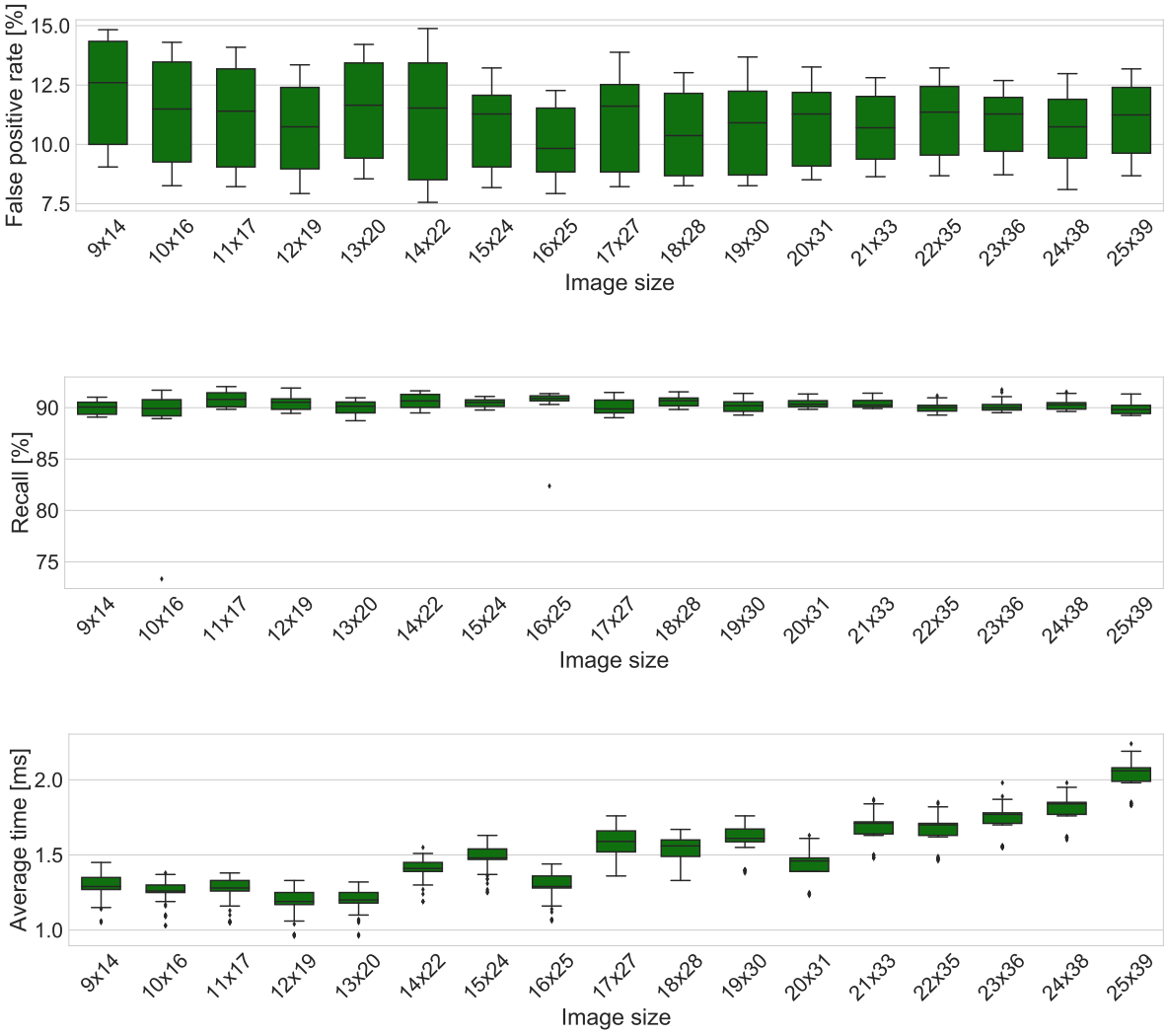


Figure 4.7: Results of all 765 tested NNs with regard to the used input size

The results with regard to the input image size is presented in figure 4.7. These results resemble what was already observed in the previous sections. The biggest influence on the classification results (FPR and recall) are caused by the amount of hidden neurons and the used activation functions. Image size however seems not to have much of an effect on those two criteria. Recall stays at about 90% for all of the different sizes. FPR also does not seem to be influenced by the image size much. It does increase a little around the very low input sizes but all together the FPR values stay in similar intervals.

The figure also shows that the input size has a large influence on the runtime. As can be seen, the time necessary to evaluate one robot candidate grows strongly with a rising amount of pixels to be evaluated. There are also a few dips visible at e.g. 16x25 and 20x31 where the time needed suddenly drops, which could be caused by how the compiler optimizes the code or how the processor handles the computation.

4.5.5 Results

The cross-validation results have shown that it is best to use either one of the three activation functions ReLU, Leaky ReLU or TanH with about a 100 neurons in the hidden layer. Since using a low resolution input image improves the runtime while not effecting the classification results by much, it is also a good choice to keep the input size low.

In order to provide a choice of actual networks which can be used on the NAO the set of tested networks was reduced to only contain those networks which are pareto-optimal[4] and fulfill the minimum requirements set in 4.2. The remaining networks are listed in A.13 sorted by their achieved score. Plots showing the results of all pareto-optimal networks can be found at A.7 and A.8.

Taking the five best scoring networks from the list of pareto-optimal networks at A.13, retraining them on the complete training and validation data and finally testing them on the test set yielded the following results:

Activation	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Score
ReLU	20	12x19	7.44	90.65	1.2	28.2433
TanH	100	11x17	7.64	92.07	1.28	25.9167
TanH	50	14x22	7.64	90.65	1.43	22.4433
Leaky ReLU	100	12x19	8.47	90.45	1.22	21.5967
Leaky ReLU	50	12x19	8.47	90.45	1.2	21.9967

Table 4: Results of evaluating the five highest scoring NNs on the test set

When using the weights and requirements as described in 4.2 the network with the best result is a network using a ReLU activation function, 20 neurons in it's hidden layer and an input size of 12x19 as marked in the table above.

Even though this network is the best scoring network the results it produces are only mediocre. A recall of about 90% is very good but it is much less important for a robot detection than the FPR and the runtime. With a measured runtime of 1.2 ms at average and 1.7 ms at maximum (A.13) the network is close to the upper bounds of viable runtimes. At about 8% the FPR is also still considerably high which could lead to problems during soccer matches.

Using the best network's measured runtimes of 1.22 ms at average and 1.70 ms at maximum with equations 2.3 and 2.4 it is possible to calculate an estimate of the overall runtime for the algorithm.

- Average time estimate $\approx$ 2 ms
- Worst-case time estimate $\approx$ 10.5 ms

The worst-case time was calculated assuming that the maximum allowed candidate count is five. For the average time an average of 1 candidate per image is assumed.

[4]With regard to this problem a network is pareto-optimal when there is no other network which improves on FPR, recall and runtime at the same time.

4.6 Convolutional Neural Networks

Due to the amount of additional influences on the classification results of CNNs as outlined in 3.5.3, it wasn't possible to train all possible combinations of the parameters that needed evaluating. So unlike the 765 tested NNs, the evaluation of CNNs required several different sets of parameter combinations. Again, first the network parameters were assessed by using the cross-validation results and then the five highest scoring parameter combinations were evaluated on the test set.

4.6.1 Evaluation of Network Structures

Since the computational resources on the NAO are very limited only small CNNs could be tested. The way this was done was to start with the smallest possible CNN-structure, a convolutional layer followed by a fully-connected layer, and then gradually enlarging the networks until they get too complex to run on the NAO.

Doing this the following structures were created:

- CF
- CAF
- CMF
- CAMF
- CFF
- CAFF
- CMFF
- CAMFF

The notation used here describes the sequence of layers used in the network architecture. Each layer is symbolized by letters as follows:

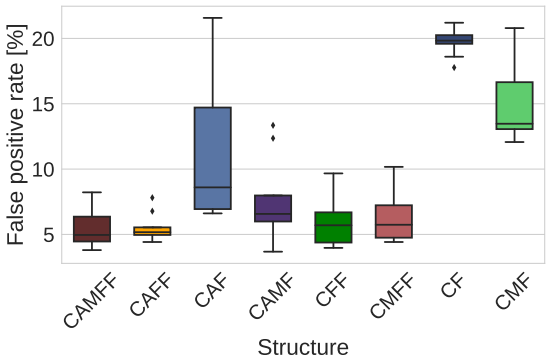
- C: Convolutional layer
- CA: Convolutional layer using an activation function (ReLU only)
- M: Max pooling layer
- F: Fully-connected layer

So a CMFF-network consists of a convolutional layer followed by a max pooling layer and two fully-connected layers.

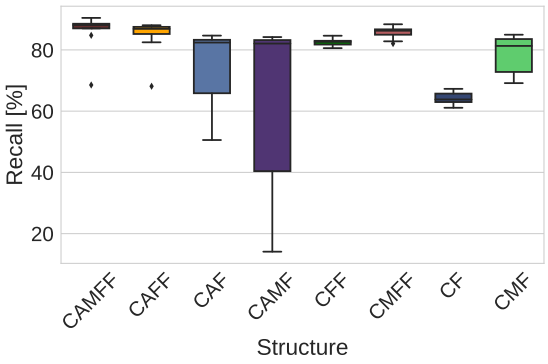
Each of these CNN-architectures was then tested on these parameter sets:

- Strides = [1]
- Amount of convolutional kernels = [1, 10, 20]
- Size of convolutional kernels = [3x3, 5x5, 7x7]
- Amount of hidden neurons = [50]
- Activation functions = [ReLU]
- Input image sizes = [18x28]

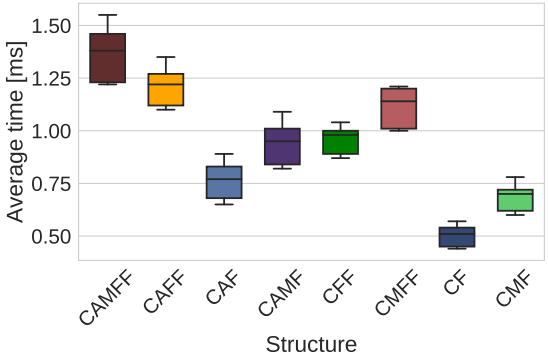
While converting the networks for the *tiny-dnn* c++ library a problem arose for the structures using max pooling layers. Networks using odd input width or height could not be converted due to different implementations in the *caffe* and *tiny-dnn* libraries. For this reason the input was chosen to be 18x28 for this analysis.



(a) FPR



(b) Recall



(c) Computation times

Figure 4.8: CNN results with regard to the used network structure

The results of the structure analysis are shown in figure 4.8. Looking at plot (a) it is visible that the CAF, CF and CMF structures lack behind the other structures in terms of FPR. The CAMF structure was able to achieve low FPR but in some cases this was coupled with a very low recall. This happens when a network tends to classify most images as background. These four structures all share the common characteristic that they lack a second fully-connected layer.

When investigating the other four architectures the classification results turn out to be much more stable. With regard to FPR they are very similar except that CFF and CMFF structures also include some lower scoring networks than CAMFF and CAFF. As can be seen in the recall plot (b) CAMFF and CAFF also achieved a slightly higher recall than CFF and CMFF. The drawback of the CAMFF and CAFF architectures becomes apparent in plot (c). Their runtime is much higher than that of CFF and CMFF, especially the CFF-structure was able to maintain low runtimes but still achieve very good classification results. This is also visible in A.15 which shows that the three highest scores were all gained by CFF networks and that it is the only structure which managed to stay within the bounds of the minimum requirements for all tested networks.

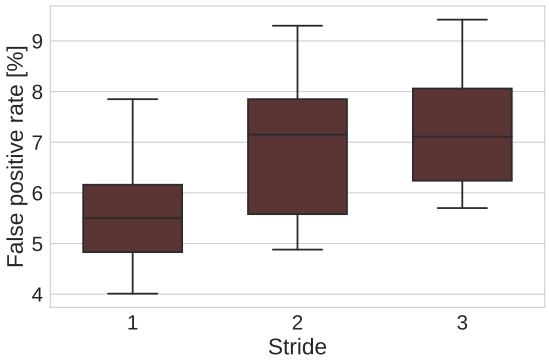
4.6.2 Evaluation of Strides

Using the CFF structure the following sets of parameters were tested:

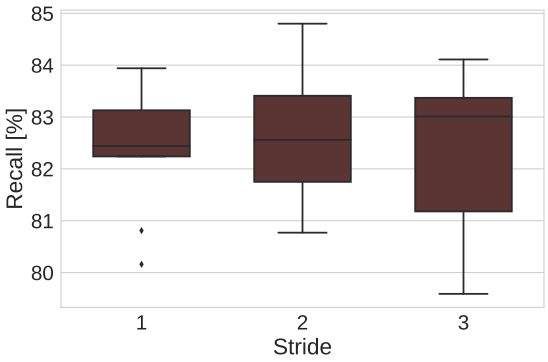
- Strides = [1, 2, 3]
- Amount of convolutional kernels = [1, 2, 5]
- Size of convolutional kernels = [3x3, 5x5, 7x7]
- Amount of hidden neurons = [50]
- Activation functions = [ReLU]
- Input image sizes = [18x28]

The results in figure 4.9 show that the FPR increases when the stride increases. This is expectable as the increase in stride means that less information is extracted from the images. It is hard to tell how exactly the stride impacts the recall, since there is no clear change visible. Looking at the median it is possible to speculate that the recall increases slightly with increasing stride but it would need more tests to validate this. Since all three strides provided good recall any stride would be an acceptable choice with regard to this criterion.

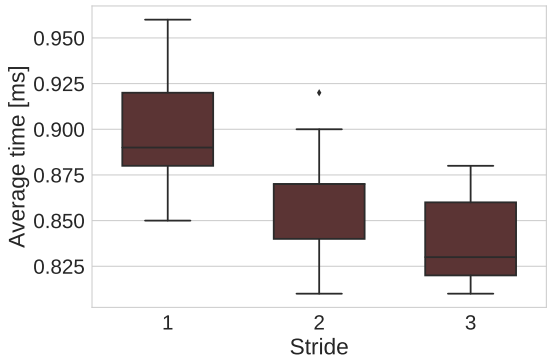
The runtime plot (c) shows that runtime gets better when the stride is increased. Since FPR is weighed higher than the runtime in the scoring function the three best scoring networks all use a stride of 1 as can be seen in A.17. Based on this, testing was continued using a stride of 1.



(a) FPR



(b) Recall



(c) Computation times

Figure 4.9: CNN results with regard to the used stride

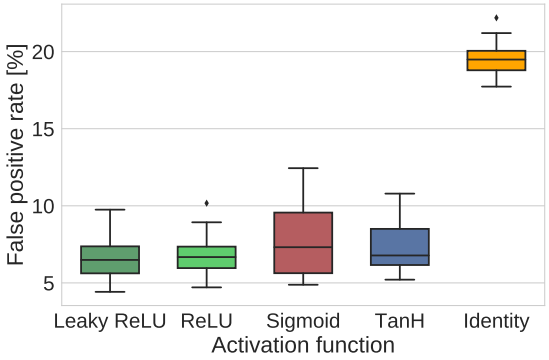
4.6.3 Evaluation of Activation Functions

For evaluating the different activation functions, combinations of the following parameter sets were tested while using a stride of 1 and the CFF network structure :

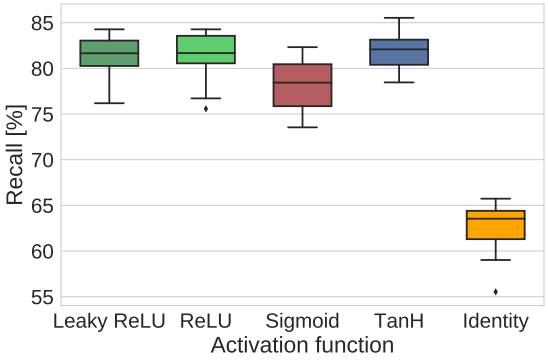
- Amount of convolutional kernels = [1, 2, 5, 10]
- Size of convolutional kernels = [3x3, 5x5, 7x7]
- Amount of hidden neurons = [10, 20, 50]
- Activation functions = [ReLU, Leaky ReLU, TanH, Sigmoid, Identity]
- Input image sizes = [17x27]

The results of the tested CNNs are shown in figure 4.10. Just like in 4.5.2 the activation functions leaky ReLU, ReLU and TanH score very similarly on all three criteria. When comparing the results of the sigmoid function here with those from 4.5.2, the function's

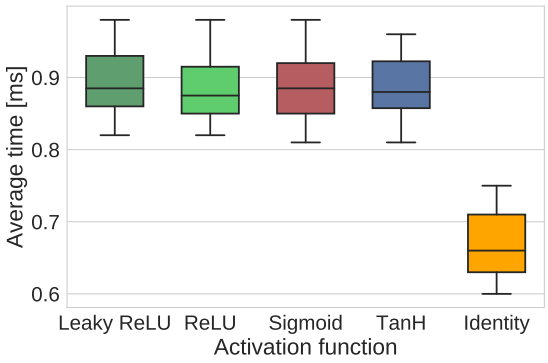
classification results are much closer to the top three functions. Even though the identity function's runtime turns out to be much better than the other's, it is evident that for CNNs it is not a viable choice due to it's low FPR and recall.



(a) FPR



(b) Recall



(c) Computation times

Figure 4.10: CNN results with regard to the used activation function

Looking at figure 4.10 and the list of networks at A.19 it can be argued that either of the functions leaky ReLU, ReLU, Sigmoid or TanH is be feasible but due to the high amount of parameters only one function could be chosen for further testing. Based on the recommendation in [10] and the fact that the highest scoring function in A.19 used a ReLU function, this function was chosen for further testing.

4.6.4 Evaluation of the Kernel Count, Kernel Size, Neuron Count and Input Dimensions

For the evaluation of the amount of convolutional kernels, their size, the hidden neuron count and the input size a large set of networks was created using the CFF structure with ReLU activation, a stride of 1 and the following sets of parameters:

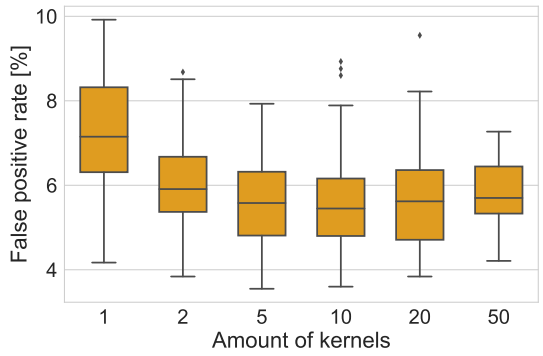
- Amount of convolutional kernels = [1, 2, 5, 10, 20, 50]
- Size of convolutional kernels = [3x3, 5x5, 7x7]
- Amount of hidden neurons = [10, 20, 50]
- Activation functions = [ReLU]
- Input image sizes = [14x22, 15x24, 16x25, 17x27, 18x28, 19x30, 20x31, 21x33, 22x35, 23x36, 24x38, 25x39]

All combinations together result in a total amount of 648 CNNs.

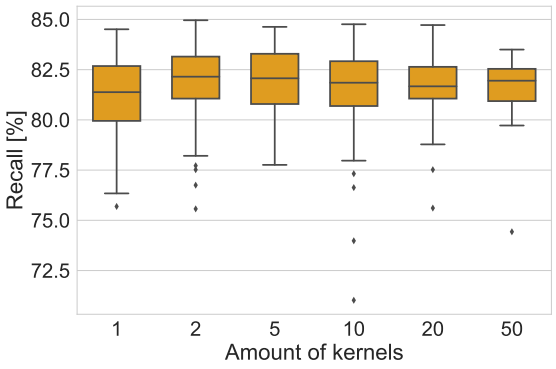
Unfortunately during evaluation the created networks showed some problems. As can be seen in A.20 the cross-validation results of the tested networks are arranged in six "steps". The reason for these steps is that some networks failed to train in such a way that they will simply label every input as background. This causes for these networks to achieve 0% FPR and 0% Recall. When now used for the cross-validation the networks which failed to train properly drag down the averaged classification results. Thus the steps in A.20 represent such combinations of network parameters which included 1, 2, 3, 4 or 5 networks which failed to train and were averaged during cross-validation. Interestingly this only happened for such combinations where the amount of kernels was much larger than the amount of hidden neurons.

In order to avoid that this problem has an effect on the following evaluations all those networks which showed that at least one of it's cross-validation elements failed to train were cut from the data. This also means that some of the combinations of the parameter sets above were not evaluated. In total 148 networks were cut such that 500 networks remained and were evaluated in the following sections. The remaining networks can be viewed at A.21. All networks that were cut are listed at A.23.

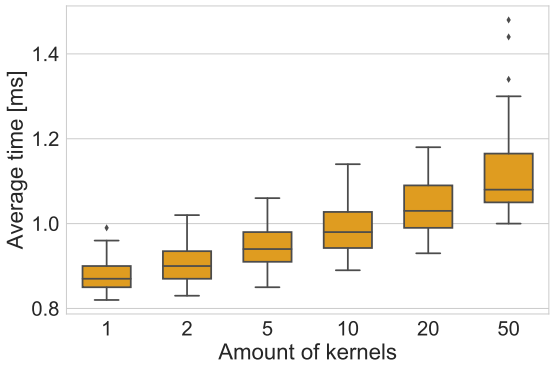
Evaluation of the Kernel Count



(a) FPR



(b) Recall



(c) Computation times

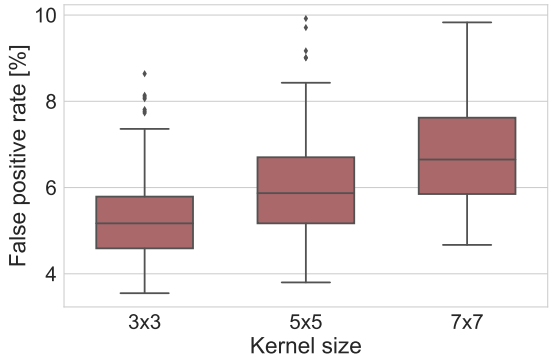
Figure 4.11: CNN results with regard to the amount of used convolutional kernels

When increasing the number of kernels the classification results of the networks tend to get better as can be seen from (a) and (b). However, starting from 20 kernels and up the results are decreasing again. Due to the large increase in network parameters when using larger amounts of kernels, the networks might start to overfit the data from 20 kernels upwards.

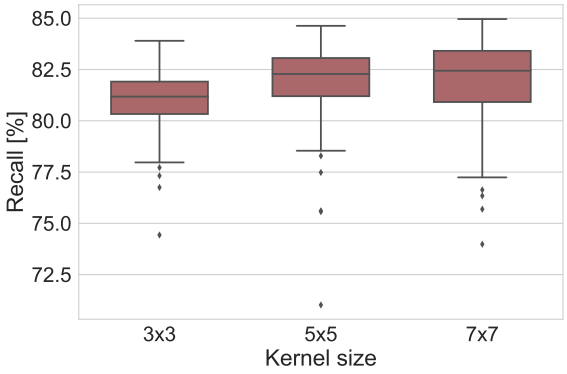
The number of kernels also has a high influence on the runtime as can be seen in (c). Increasing the kernel count quickly leads to a long runtime which suggests to keep the kernel amount as low as the classification results allow it.

Since the classification results decreased when using more then 10 kernels and the runtime is also highly dependent on this parameter, the amount of kernels should be kept at 10 or below. Looking at the list of networks in A.24 supports this suggestion as well. The highest scores were achieved when using a kernel count of 1 to 10.

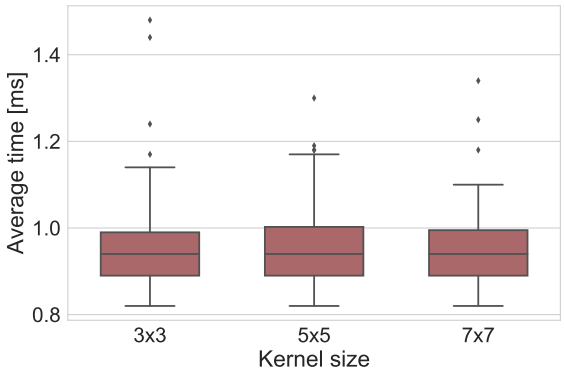
Evaluation of the Kernel Size



(a) FPR



(b) Recall



(c) Computation times

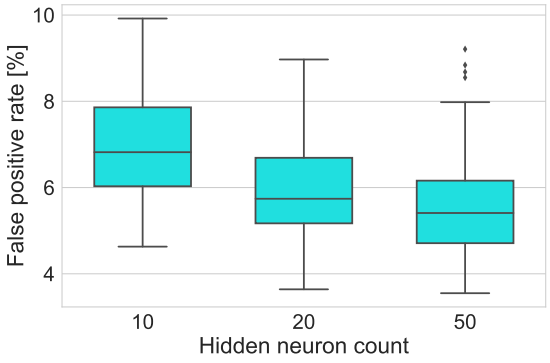
Figure 4.12: CNN results with regard to the used kernel size

As shown in figure 4.12 (a) the FPR worsens with rising kernel sizes. The recall however rises slightly with larger kernel sizes.

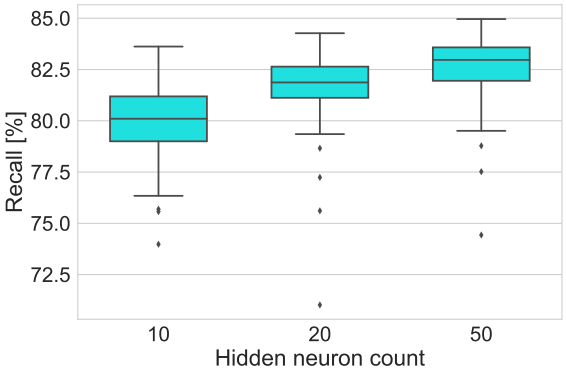
The runtime of the network seems not to be significantly influenced by the kernel size.

Looking at the results of the tested networks in A.24 shows that most of the better scoring networks use 3x3 or 5x5 kernels. 7x7 kernels do provide some good results as well but generally score lower due to lower FPRs.

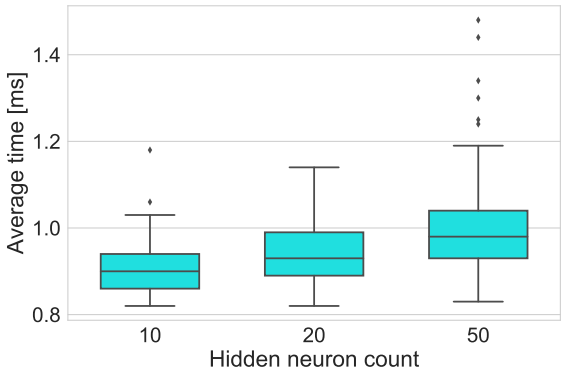
Evaluation of the Neuron Count



(a) FPR



(b) Recall



(c) Computation times

Figure 4.13: CNN results with regard to the used neuron count

Increasing the hidden neuron count also has a positive effect on both FPR and recall.

The runtime increases with the neuron count. Even though the impact on the runtime is high, the influence on the classification results proves to outweigh this for the tested neuron counts. In general the higher neuron counts achieved better scores as visible in A.24.

Evaluation of the Input Dimensions

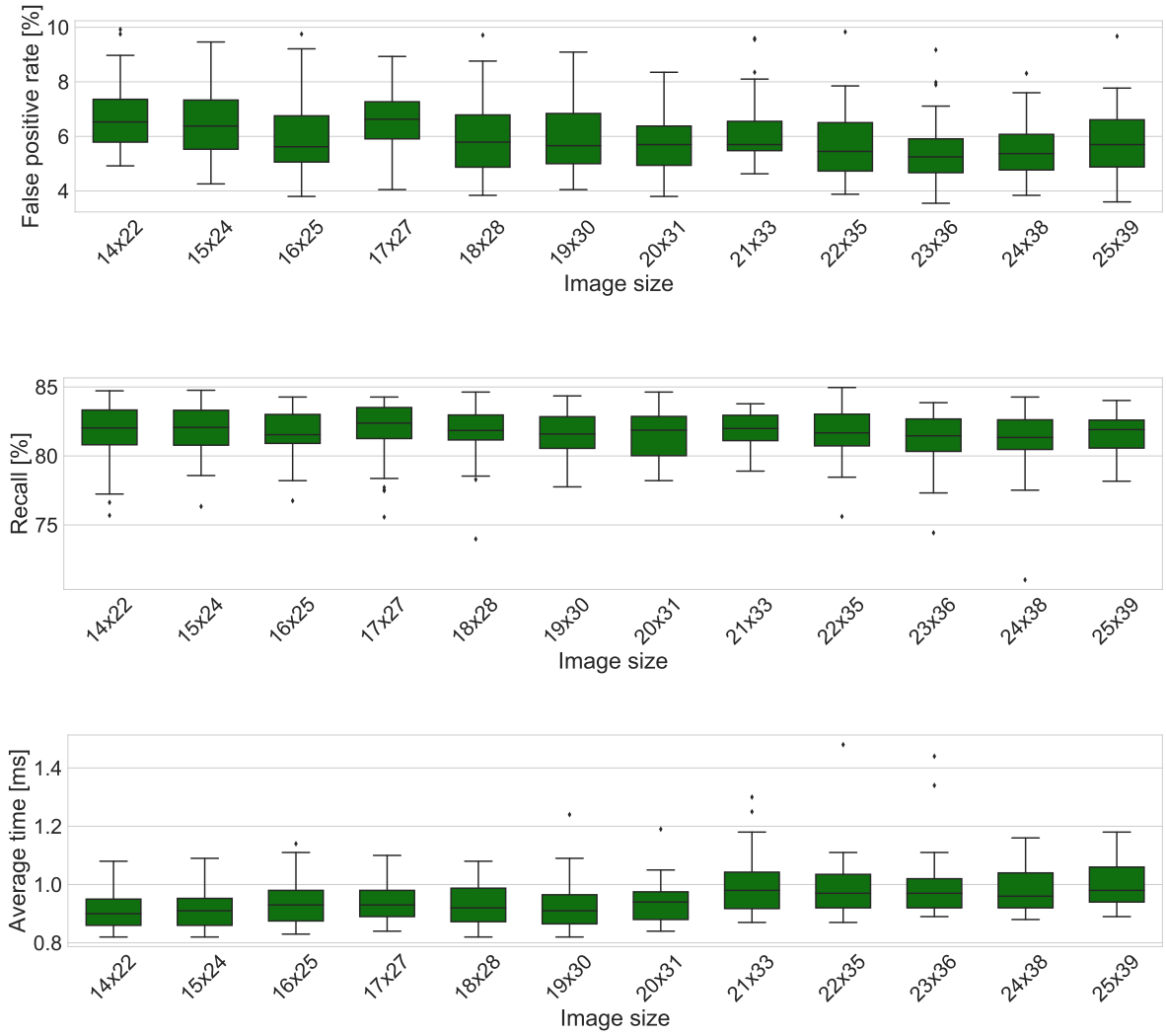


Figure 4.14: CNN results with regard to the used image size

Figure 4.14 presents the results of the networks with regard to the input dimensions. As visible in the first plot the FPR slightly decreases with rising input dimensions. This can be credited to the increased information contained in larger images. The recall stays the same for almost all image sizes.

The input dimensions also have an influence on the runtime. When increasing the input dimensions the runtime also increases. This influence is however not as big as the ones caused by the amount of kernels or the neuron count.

The results in A.24 show that there is no single best input size. In combination with other parameters all of the tested sizes were able to achieve good scores.

4.6.5 Results

Summarizing the last sections the following conclusions can be drawn about the different parameters.

- Of the tested CNN-architectures the CFF structure yields the best compromise between good classification results and runtime.
- A stride of 1 proved to yield the highest scores. If the runtime's influence on the score was weighed higher, larger strides would become more viable too.
- Either leaky ReLU, ReLU, Sigmoid or TanH can be used as activation function.
- The amount of kernels should be kept low due to its high impact on runtime and the probability of overfitting. Preferably it should be kept between 1 and 10.
- The 3x3 kernel size achieved the best results when compared to 5x5 and 7x7.
- A higher number of hidden neurons promises better classification results. This will have it's limit but from the tested set 50 neurons provided the best results.
- It was not possible to find a rule for the input size. Both high and low sizes were able to score well.

As can be seen in A.24, most of these conclusions are also represented by the five highest scoring networks. All of these networks use a number of kernels smaller or equal to 10. Four out of five use a 3x3 kernel. All five use 50 neurons and both high and low input dimensions were able to achieve high scores.

The results of the evaluation of these five networks on the test set is shown in table 5.

Act.	Kernels	K. Size	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Score
ReLU	5	3	50	23x36	3.51	86.38	0.99	54.6
ReLU	2	3	50	24x38	3.93	87.6	0.95	53.28
ReLU	2	3	50	18x28	4.96	87.8	0.89	48.37
ReLU	10	5	50	16x25	4.75	88.21	0.98	47.97
ReLU	5	3	50	16x25	6.2	88.21	0.92	40.47

Table 5: Results of evaluating the five highest scoring CNNs on the test set

While the first network in the table provides the lowest FPR and the highest overall score, the third network might be a good choice too if the first proves to be too slow, especially due to it's lower maximum runtime measured at 1.29 ms (A.24).

Comparing the results on the test set in table 5 and the cross-validation results in A.24 shows that the first two networks from table 5 were able to improve their score by a general improvement in recall. The last three networks however all scored worse on the test set due to a drop in FPR. All five results show that the networks generalize well enough to be able to classify previously unseen data.

Using the best network's measured runtimes of 0.99 ms at average and 1.66 ms at maximum (A.24) and equations 2.3 and 2.4 it is possible to estimate the overall runtime for the algorithm.

- Average time estimate: ca. 1.7 ms
- Worst-case time estimate: ca. 10.4 ms

Like with the feature-based classification, the worst-case time was calculated assuming that the maximum allowed candidate count is five. For the average time an average of 1 candidate per image was assumed.

5 Conclusion and Outlook

The goal of this thesis was to find an implementable solution for a robot detection on the NAO robotic system. While comparing the two approaches of feature-based NN and CNN classification it was possible to observe that both provide a feasible solution for implementation on the NAO.

Activation	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Score
ReLU	20	12x19	7.44	90.65	1.2	28.2433
TanH	100	11x17	7.64	92.07	1.28	25.9167
TanH	50	14x22	7.64	90.65	1.43	22.4433
Leaky ReLU	100	12x19	8.47	90.45	1.22	21.5967
Leaky ReLU	50	12x19	8.47	90.45	1.2	21.9967

Table 6: Results of evaluating the five highest scoring NNs on the test set

Act.	Kernels	K. Size	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Score
ReLU	5	3	50	23x36	3.51	86.38	0.99	54.6
ReLU	2	3	50	24x38	3.93	87.6	0.95	53.2867
ReLU	2	3	50	18x28	4.96	87.8	0.89	48.3733
ReLU	10	5	50	16x25	4.75	88.21	0.98	47.97
ReLU	5	3	50	16x25	6.2	88.21	0.92	40.47

Table 7: Results of evaluating the five highest scoring CNNs on the test set

Looking at table 6 and 7 it is however apparent that the results achieved by the feature-based approach provided much lower scores than the CNN ones. One problem with the feature-based approach is that the features have to be created and evaluated manually. The CNNs can simply learn them during their training. The results by no means show that the CNN approach has to be the better method. Since the basic NNs are much faster than the CNNs when disregarding the feature-calculation time, finding the correct features which produce good classification results while staying low on computational cost could show that the feature-based approach is better after all. With the features found and tested and the evaluation criteria used during this thesis however, CNNs were proven to be superior in terms of both runtime and classification outcome.

The CNN results are likely to be improvable as well. It is not possible to rule out that the classification approaches are not overfitting the given data at all. Using methods like regularization and dropout could further improve the classification results. The parameter search could also be extended to find parameter combinations which work even better than the ones tested.

With the classification methods provided by this thesis the next step that has to be taken is to actually make use of the detected robots. So far, there is no means to calculate the position of the detected robots on the field. Also, playing strategies which make use of the new information have to be implemented. Another important point is that this detection

does not separate between friend and foe, so a jersey detection is another necessary part to make full use of the provided robot detection.

Another factor worth reconsidering is the candidate generation implemented during this thesis. While it provides reasonable results it still offers much room for improvement. Especially the dependency on the field color detection is a big problem because it makes the detection prone to produce erroneous results when the playing conditions are not optimal.

In summary this thesis proves that both the feature-based approach and the CNNs are viable solutions for a successful robot detection on the NAOs. It provides the basis for further examinations and improvements through the evaluated network parameters and by supplying tools like automated creation and testing of neural networks. However, there are still many things that have to be done before the robot detection can be used to it's full extend during RoboCup games.

References

[1] Aldebaran, *Nao - video camera.* http://doc.aldebaran.com/2-1/family/robots/video_robot.html. Last visited: 15.03.2017.

[2] E. Alpaydin, *Introduction to machine learning*, Adaptive computation and machine learning, The MIT Press, Cambridge, Massachusetts and London, England, third edition (online-edition) ed., 2014.

[3] B-Human Team, *Team report and code release 2016.* https://github.com/bhuman/BHumanCodeRelease/raw/master/CodeRelease2016.pdf, 2016. Last visited: 17.03.2016.

[4] Jaiden Ashmore, *Robot Detection Using Bayesian Machine Learning*, thesis, The University of New South Wales, Sydney, Australia, 25.08.2014.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, *Caffe: Convolutional architecture for fast feature embedding*, 2014.

[6] Jimmy Kurniawan, *Multi-modal Machine-learned Robot Detection for RoboCup SPL*, PhD thesis, The University of New South Wales, Sydney, Australia, 24.08.2011.

[7] J. Kacprzyk, S. Gunn, I. Guyon, M. Nikravesh, and L. A. Zadeh, *Feature extraction: Foundations and applications*, vol. 207 of Studies in Fuzziness and Soft Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[8] Kevin Markham, *Simple guide to confusion matrix terminology.* http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/, 2014. Last visited: 18.03.2017.

[9] A. Krizhevsky, Sutskever Ilya, and Hinton Geoffrey, *Imagenet classification with deep convolutional neural networks.* https://www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf, 2012. Last visisted: 23.03.2017.

[10] F.-F. Li, Karpathy Andrej, and J. Johnson, *Cs231n: Convolutional neural networks for visual recognition.* http://cs231n.github.io/neural-networks-1/, 2016. Last visited: 21.03.2017.

[11] Martin Engel, *Anwendung von maschinellem Lernen zur echtzeitfähigen und kalibrierungsfreien Erkennung von humanoiden Fußballrobotern*, bachelor thesis, HTWK Leipzig, 08.08.2012.

[12] Nathapon Olaf Lüders, *Object Localization on the Nao Robotic System Using a Deep Convolutional Neural Network and an Image Contrast Based Approach*, PhD thesis, Technical University of Hamburg, Hamburg, 15.08.2016.

[13] E.-I. Osawa, H. Kitano, M. Asada, Y. Kuniyoshi, and I. Noda, *Robocup: the robot world cup initiative*.

[14] Pascal Loth, *Implementierung und evaluation einer robusten echtzeitkantendetektion auf dem humanoiden nao-robotiksystem: Bachelor thesis*. http://www.hulks.de/_files/BA_Pascal-Loth.pdf, 19.10.2015. Last visited: 02.04.2017.

[15] Pedregosa et al., *Scikit-learn: Machine learning in python*, 2011.

[16] F. Poppinga, *Implementation and Evaluation of Audio Based Methods for Robust Inter-Robot Communication*, project thesis, Helmut Schmidt Universität, Hamburg, 29.07.2016.

[17] RoboCup Federation, *A brief history of robocup*. http://www.robocup.org/a_brief_history_of_robocup. Last visited: 15.03.2017.

[18] RoboCup Technical Committee, *Robocup standard platform league (nao) rule book*. http://www.tzi.de/spl/pub/Website/Downloads/Rules2017.pdf, 01.11.2016. Last visited: 15.03.2017.

[19] P. Ross, *Alphago wins final game in match against champion go player*. http://spectrum.ieee.org/tech-talk/computing/networks/alphago-wins-match-against-top-go-player, 2016. Last visisted: 22.03.2017.

[20] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*, Prentice-Hall series in artificial intelligence, Pearson, Boston, 3. ed., internat. ed. ed., 2010.

[21] T. Schattschneider, *Formbasierte ballerkennung in echtzeit auf dem humanoiden nao-robotiksystem: Bachelor thesis*. http://www.hulks.de/_files/BA_Thomas-Schattschneider.pdf, 23.11.2015. Last visited: 02.04.2017.

[22] M. E. Schröder, *Machine learning in context of robot soccer on the humanoid nao robotic system: Project thesis*. http://www.hulks.de/_files/PA_Erik-Schr%C3%B6der.pdf, 29.01.2016. Last visisted: 02.04.2017.

[23] Softbank Robotics, *Nao website*. https://www.ald.softbankrobotics.com/en/cool-robots/nao. Last visited: 15.03.2017.

[24] Softbank Robotics Corp., *Nao datasheet*. https://www.ald.softbankrobotics.com/sites/aldebaran/files/nao_datasheet.pdf. Last visited: 15.03.2017.

[25] Tiny-dnn development team, *Header only, dependency-free deep learning framework in c++11*. https://github.com/tiny-dnn/tiny-dnn. Last visited: 21.03.2017.

[26] Unknown, *Nvidia ces 2016 press conference - slides*. https://www.slideshare.net/NVIDIA/nvidia-ces-2016-press-conference. 22.03.2017.

[27] H. Wickham and Stryjewski Lisa, *40 years of boxplots*. http://vita.had.co.nz/papers/boxplots.pdf, 2011. Last visisted: 02.04.2017.

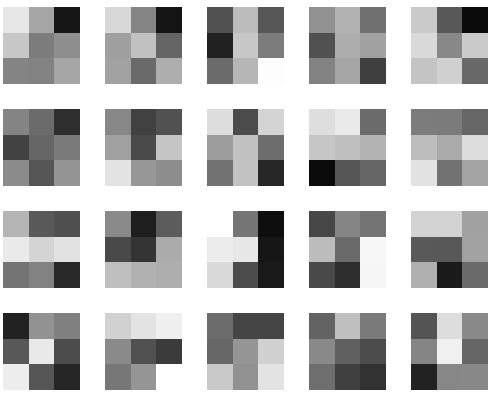
[28] A. Zhang, B. Yang, and L. Huang, *Feature extraction of eeg signals using power spectral entropy*, in 2008 International Conference on BioMedical Engineering and Informatics, IEEE, 2008, pp. 435–439.

A Appendix

A.1 Configuration File for Candidate Generation Algorithm

```
{
    "max_region_grow_factor" : 4,
    "max_region_shrink_factor" : 4,
    "min_region_length" : 10,
    "min_robot_width" : 30,
    "min_robot_height" : 40,
    "max_robot_width" : 400,
    "max_robot_height" : 480,
    "max_variance" : 5,
    "width_tolerance_factor" : 1.1,
    "height_tolerance_factor" : 4,
    "max_amount_candidates" : 5,
    "candidate_width" : 18,
    "candidate_height" : 24
}
```

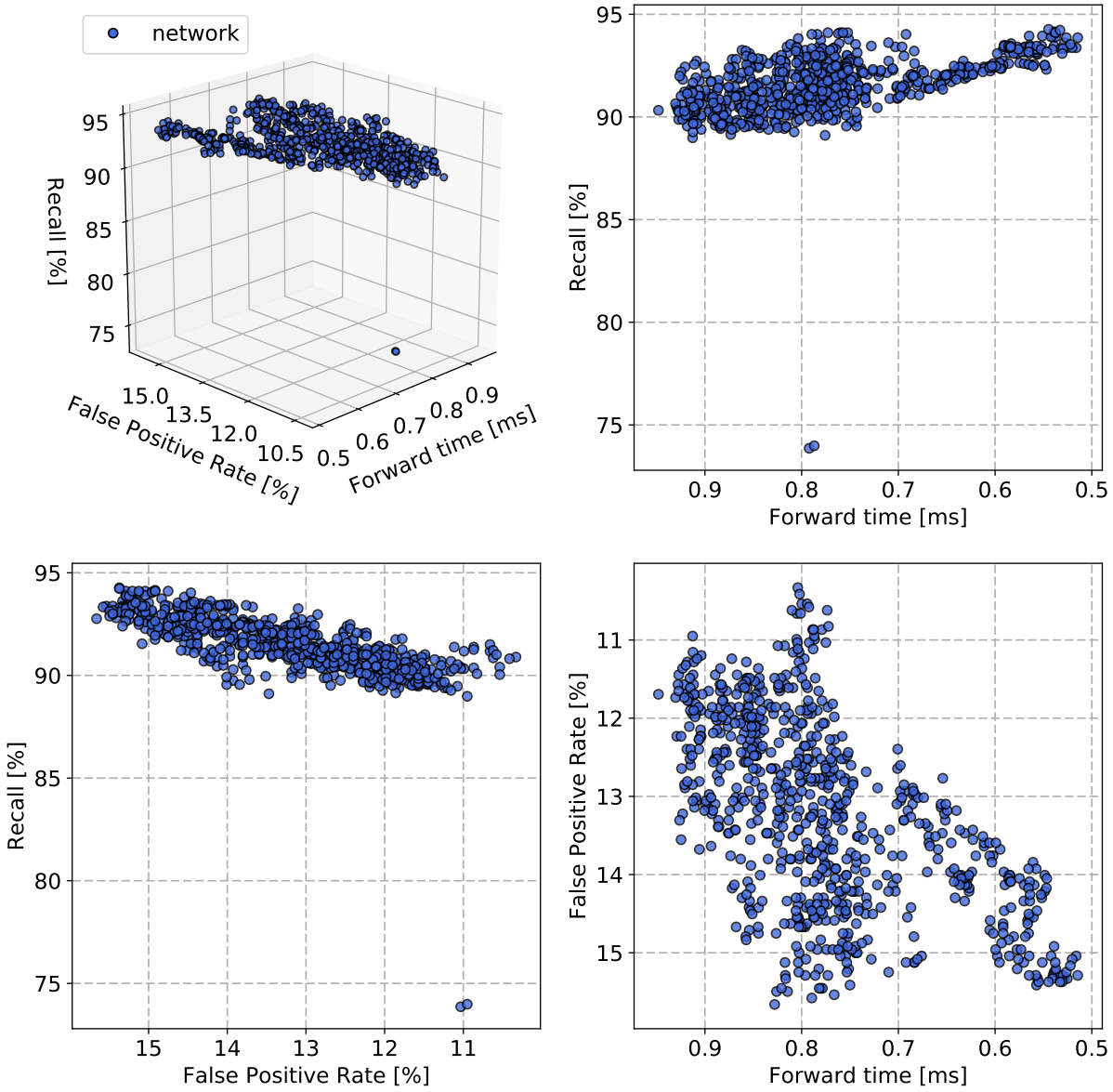
A.2 Weight visualization of a convolutional layer using 20 kernels of size 3x



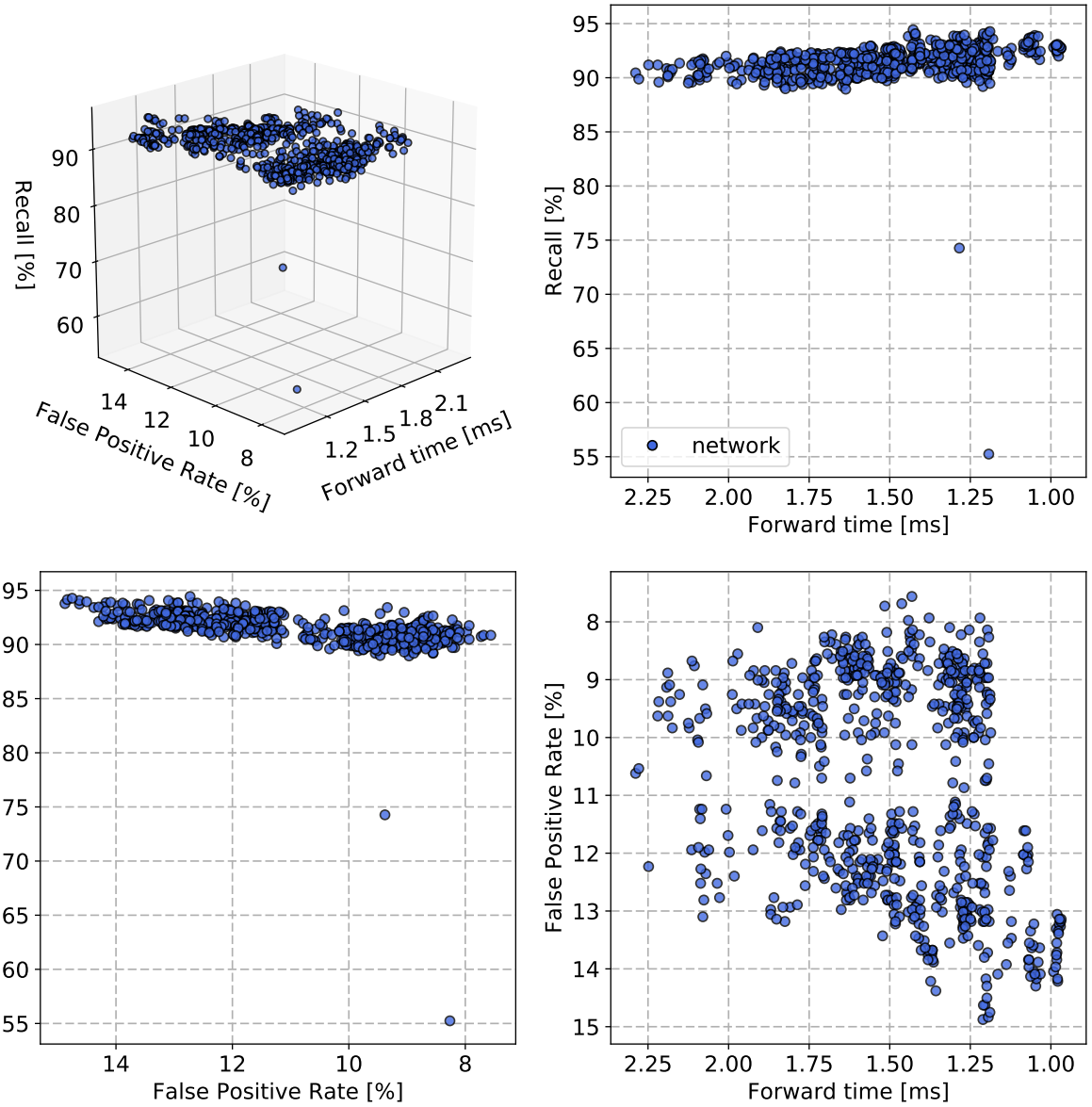
A.3 Example of a JSON file containing parameters for feature standardization


```
{
    "train_params": {
        "params": {
            "with_mean": true,
            "with_std": true,
            "copy": true,
            "mean_": [
                292.2449630369634,
                0.00019077969226122618,
                327.86514865674934,
                0.00016642968873380106,
                773447.6348630529
            ],
            "n_samples_seen_": 3003,
            "var_": [
                58272.80106754024,
                1.950751976089696e-08,
                45365.95210733148,
                1.3941424456922264e-08,
                109856991651.29988
            ],
            "scale_": [
                241.3975995480076,
                0.00013966932290555774,
                212.9928452022074,
                0.00011807380936059556,
                331446.81572056154
            ]
    }
}
```

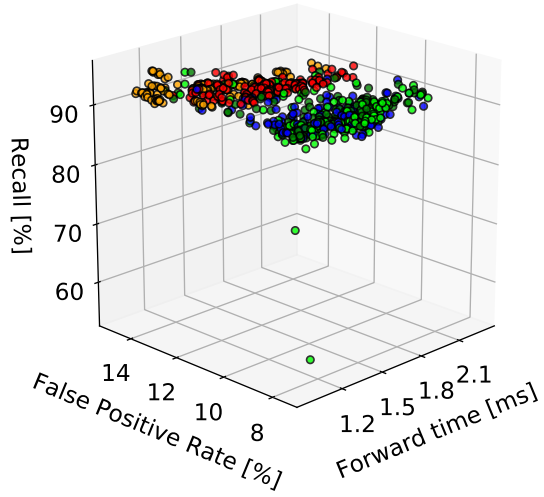
A.4 Cross-validation results of all tested Neural Networks using Feature Combination 1

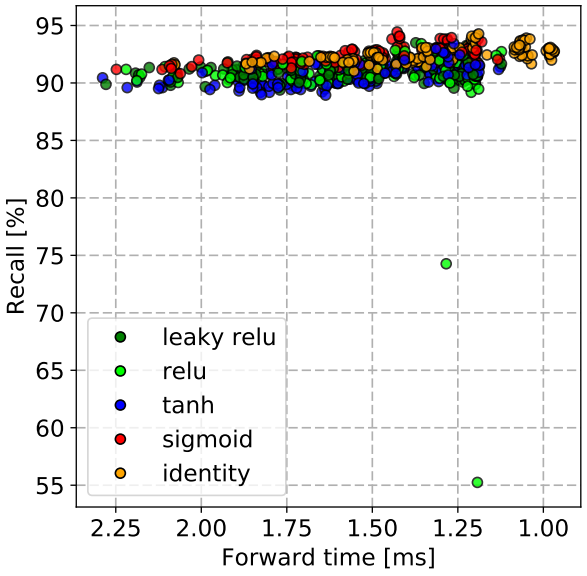


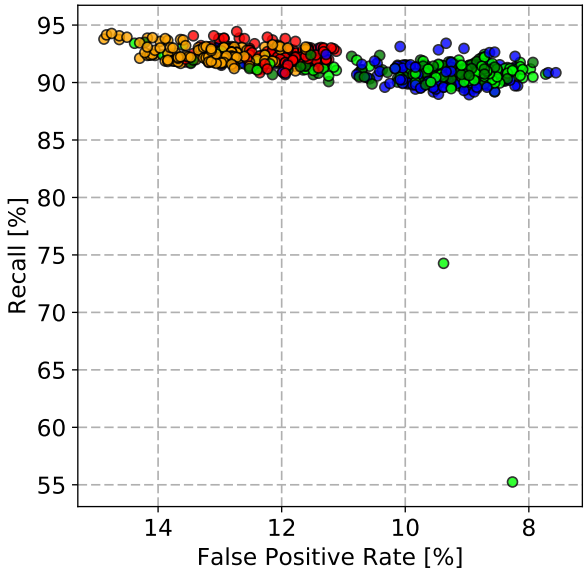
A.5 Cross-validation results of all tested Neural Networks using Feature Combination 2

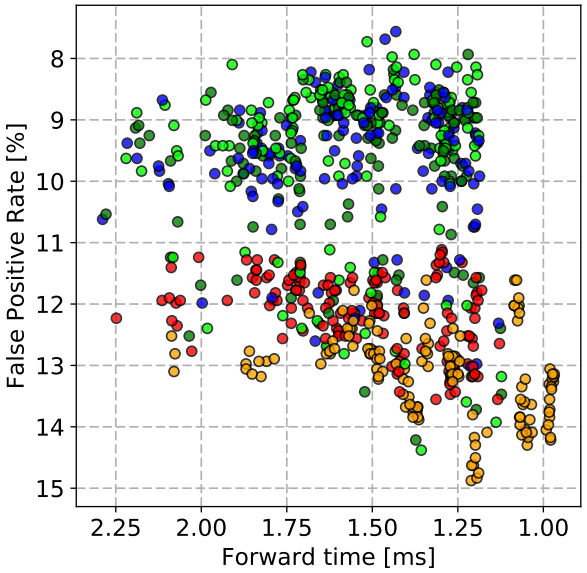


A.6 Cross-validation results of all tested Neural Networks colored by Activation Function

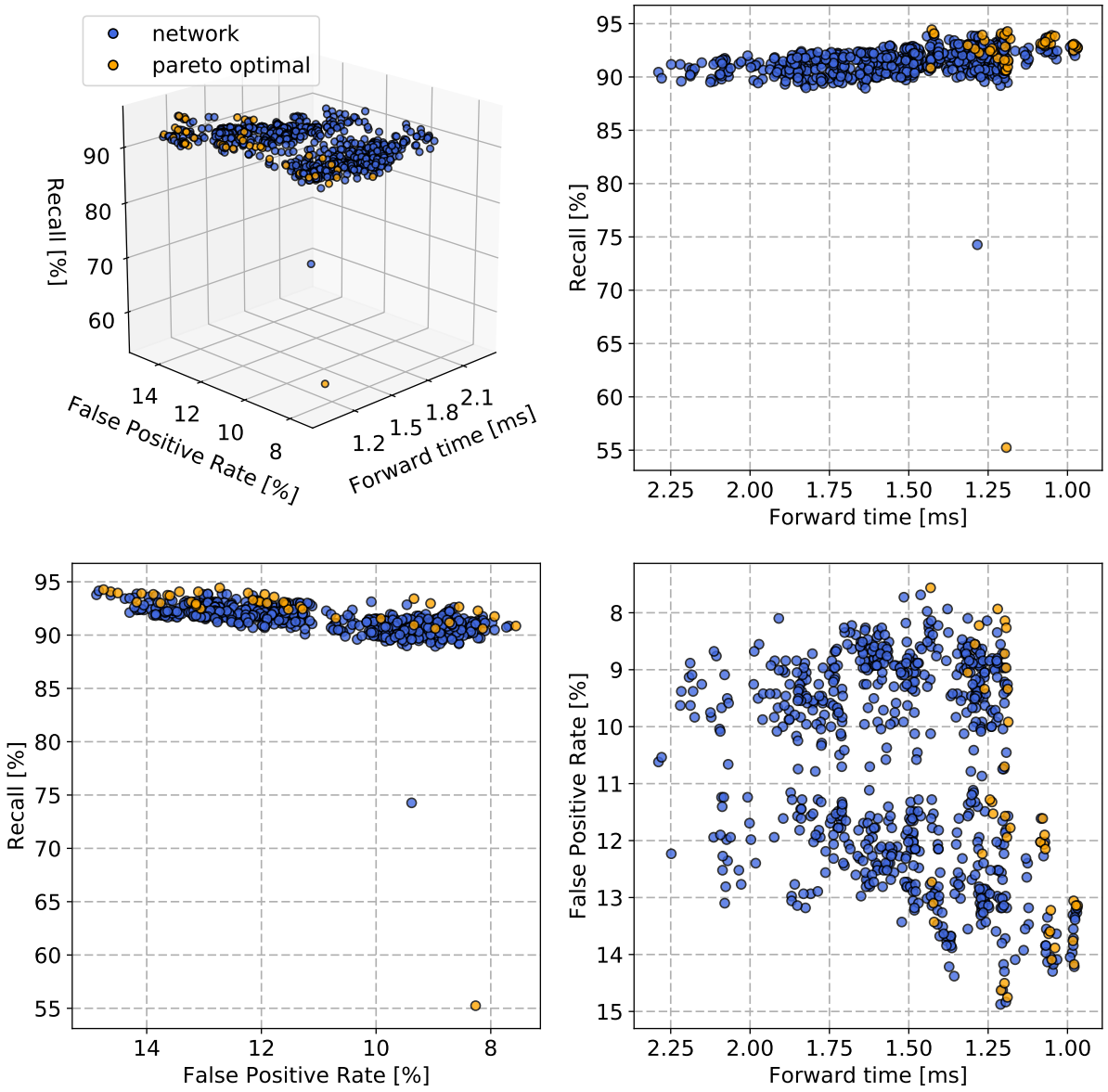




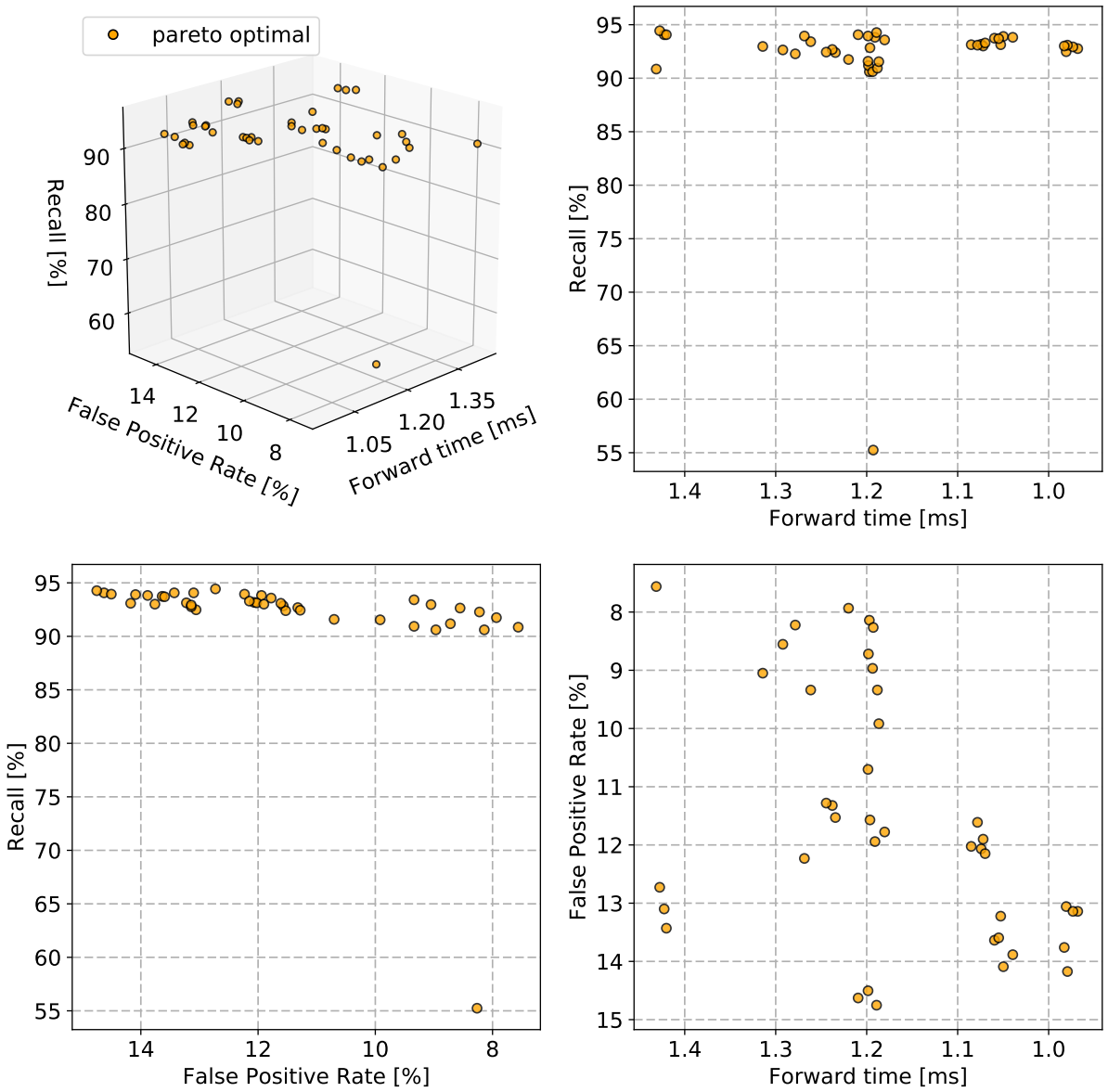




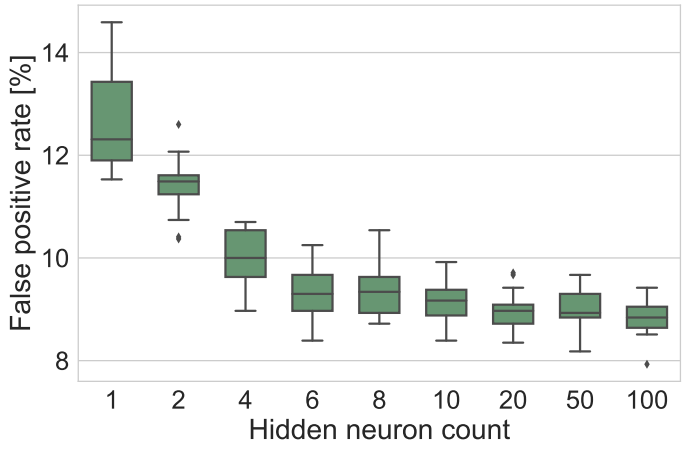
A.7 Cross-validation results of all tested Neural Networks with colored pareto-optimal Networks

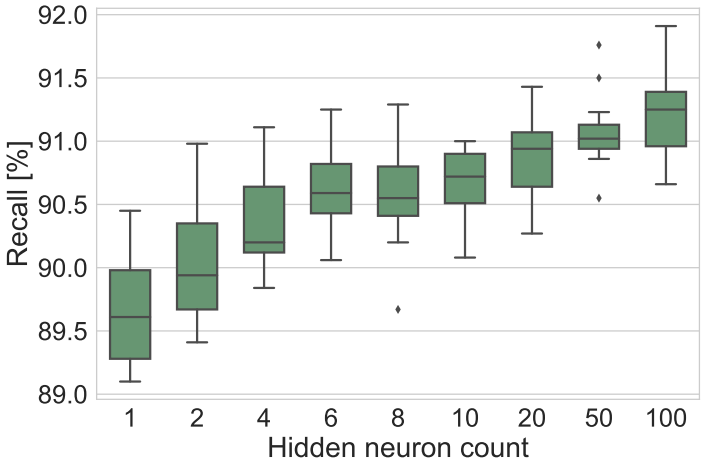


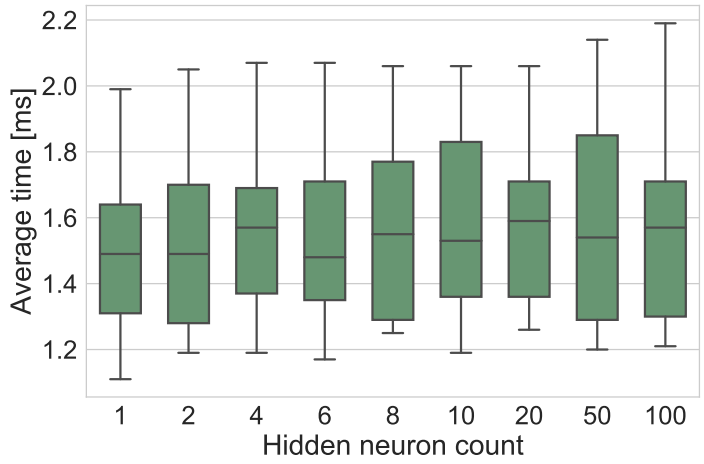
A.8 Cross-validation results of all pareto-optimal Networks



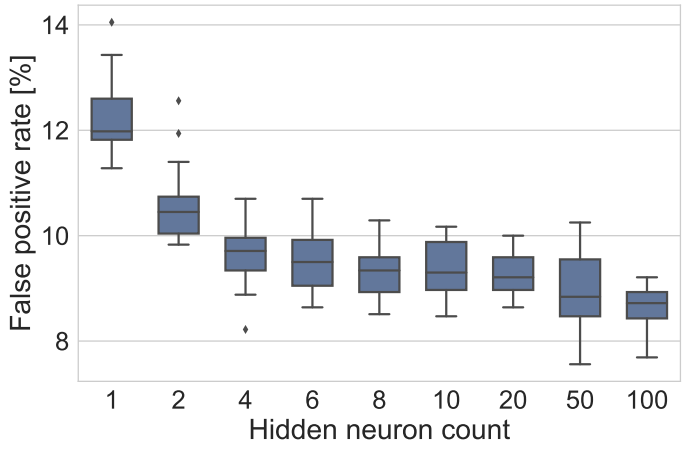
A.9 Cross-validation results with regard to the used Neuron Count for NNs using Leaky ReLU Activation

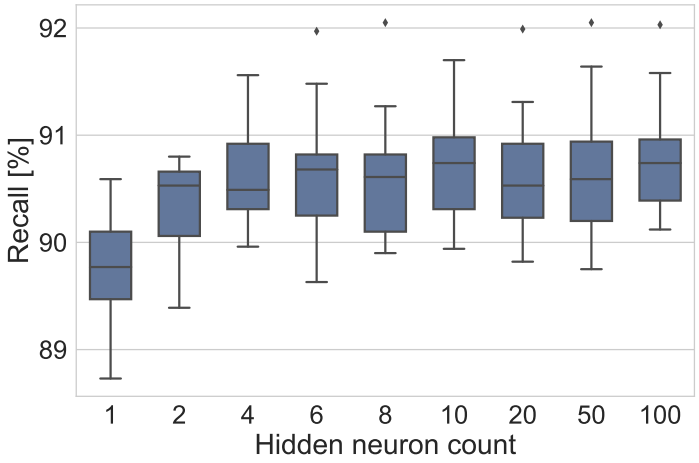


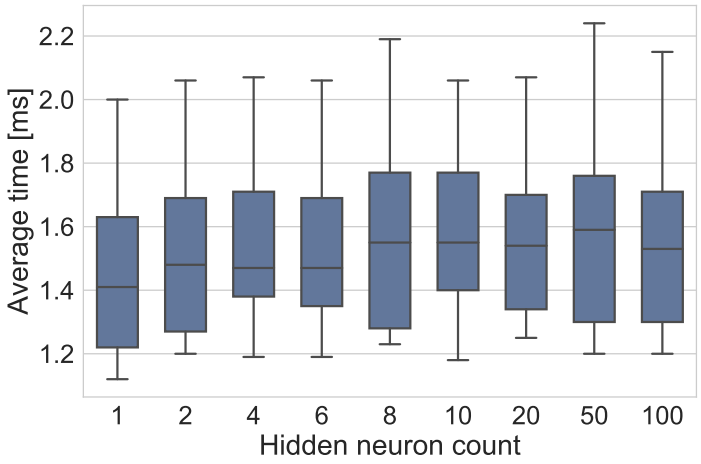




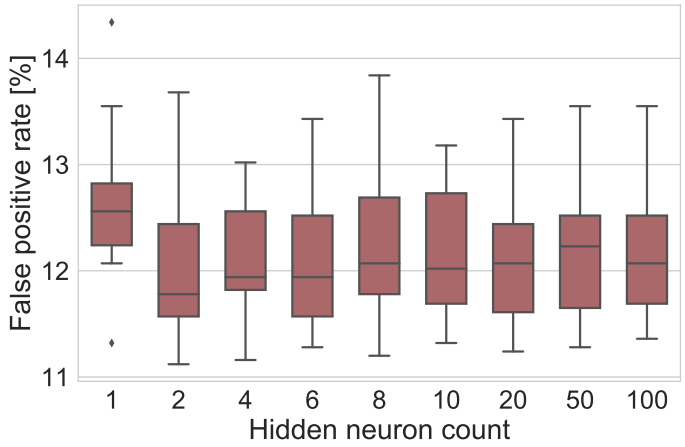
A.10 Cross-validation results with regard to the used Neuron Count for NNs using TanH Activation

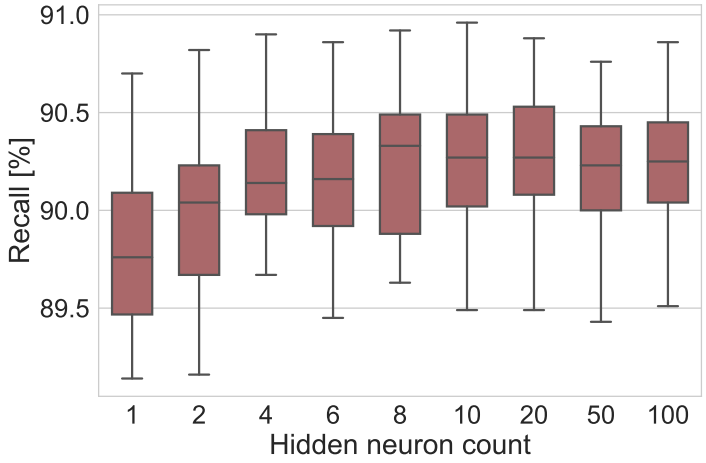


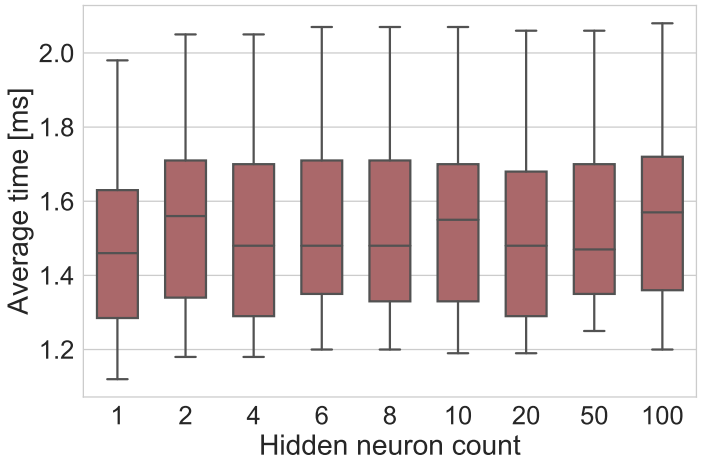




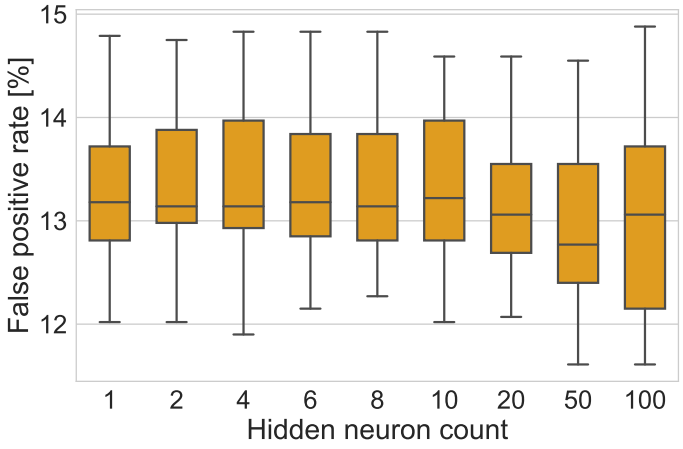
A.11 Cross-validation results with regard to the used Neuron Count for NNs using Sigmoid Activation

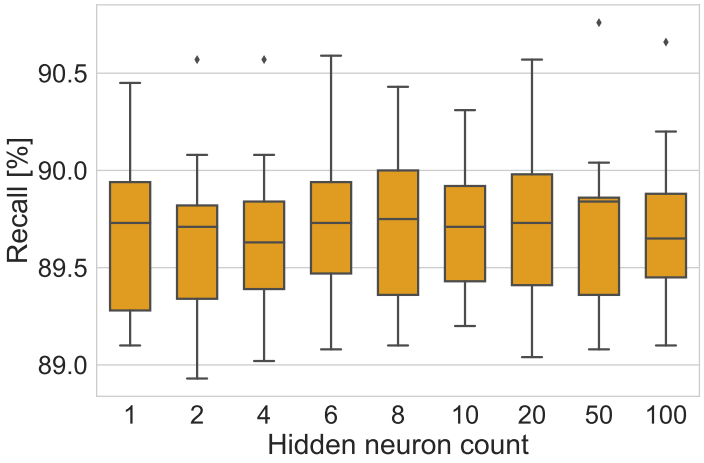


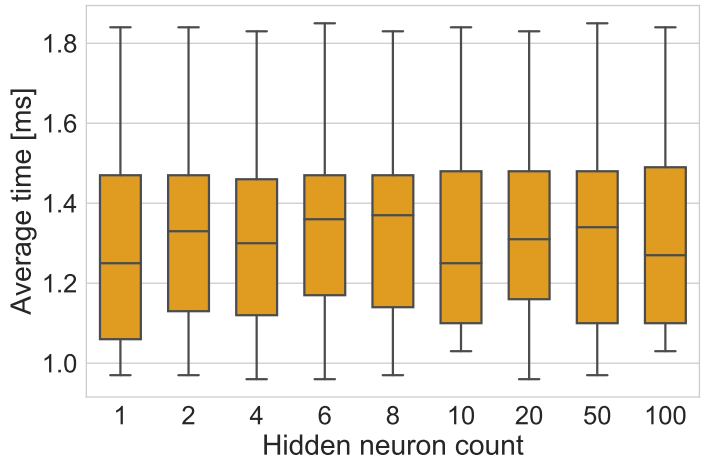




A.12 Cross-validation results with regard to the used Neuron Count for NNs using Identity Activation



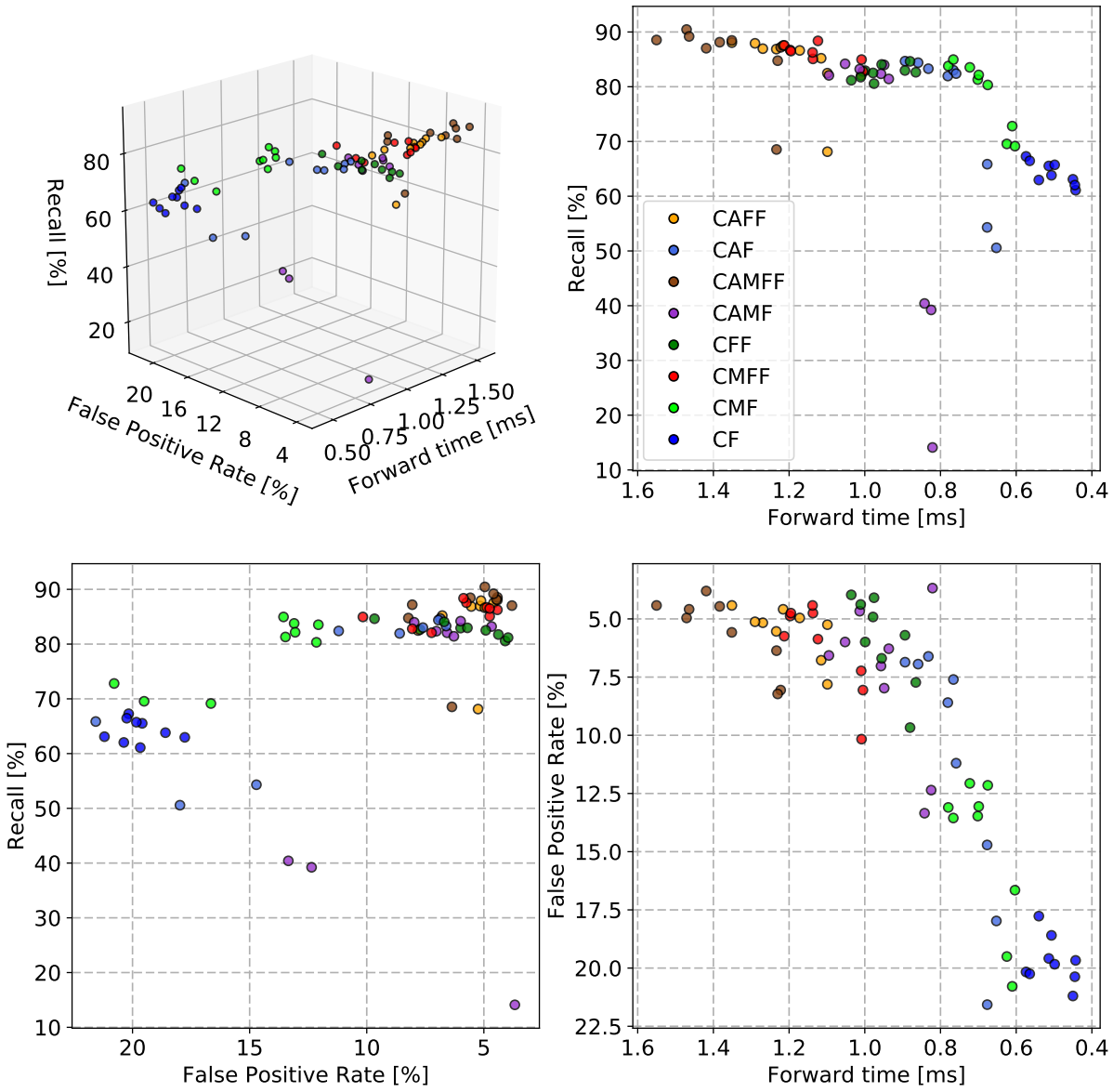




A.13 List of all pareto-optimal Neural Networks

Activation	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Max. time[ms]	Score
Leaky ReLU	100	12x19	7.93	91.75	1.22	1.70	25.27
ReLU	20	12x19	8.14	90.61	1.20	1.70	24.03
TanH	50	14x22	7.56	90.85	1.43	2.50	22.99
TanH	100	11x17	8.22	92.28	1.28	1.74	22.50
Leaky ReLU	50	12x19	8.72	91.18	1.20	1.64	20.74
TanH	50	11x17	8.55	92.64	1.29	1.85	20.45
Leaky ReLU	6	12x19	8.97	90.61	1.19	1.97	19.25
TanH	6	11x17	9.05	92.97	1.31	1.59	17.16
TanH	6	12x19	9.34	90.93	1.19	1.52	17.14
TanH	8	11x17	9.34	93.41	1.26	1.75	16.56
TanH	6	13x20	9.92	91.54	1.19	1.62	13.86
ReLU	1	10x16	8.26	55.24	1.19	1.60	0
Identity	10	11x17	14.09	93.90	1.05	1.58	0
Identity	1	11x17	13.64	93.74	1.06	1.54	0
Identity	20	11x17	13.22	93.13	1.05	1.43	0
Sigmoid	20	11x17	12.23	93.94	1.27	1.93	0
Identity	2	11x17	13.60	93.70	1.05	1.67	0
Identity	8	11x17	13.88	93.82	1.04	1.39	0
Sigmoid	10	12x19	11.94	93.82	1.19	1.74	0
Identity	2	12x19	13.06	92.48	0.98	1.38	0
Identity	50	12x19	13.14	92.76	0.97	1.36	0
Sigmoid	6	12x19	11.57	92.85	1.20	2.00	0
Identity	8	12x19	13.14	92.93	0.97	1.36	0
Sigmoid	8	12x19	11.78	93.58	1.18	1.66	0
Identity	4	13x20	14.17	93.09	0.98	1.72	0
TanH	4	13x20	10.70	91.59	1.20	1.62	0
Identity	6	13x20	13.76	93.01	0.98	1.37	0
Sigmoid	100	14x22	13.10	94.07	1.42	2.12	0
Sigmoid	10	14x22	12.73	94.43	1.43	2.00	0
Identity	1	14x22	14.63	94.07	1.21	1.88	0
Sigmoid	20	14x22	13.43	94.07	1.42	1.89	0
Identity	2	14x22	14.50	93.94	1.20	1.71	0
Identity	8	14x22	14.75	94.27	1.19	1.78	0
Leaky ReLU	1	16x25	11.53	92.40	1.23	1.98	0
Sigmoid	1	16x25	11.32	92.68	1.24	1.76	0
TanH	1	16x25	11.28	92.44	1.24	1.66	0
Identity	20	16x25	12.07	93.17	1.07	1.68	0
Identity	2	16x25	12.02	93.13	1.09	1.70	0
Identity	4	16x25	11.90	93.01	1.07	1.69	0
Identity	50	16x25	11.61	93.09	1.08	1.59	0
Identity	6	16x25	12.15	93.29	1.07	1.47	0

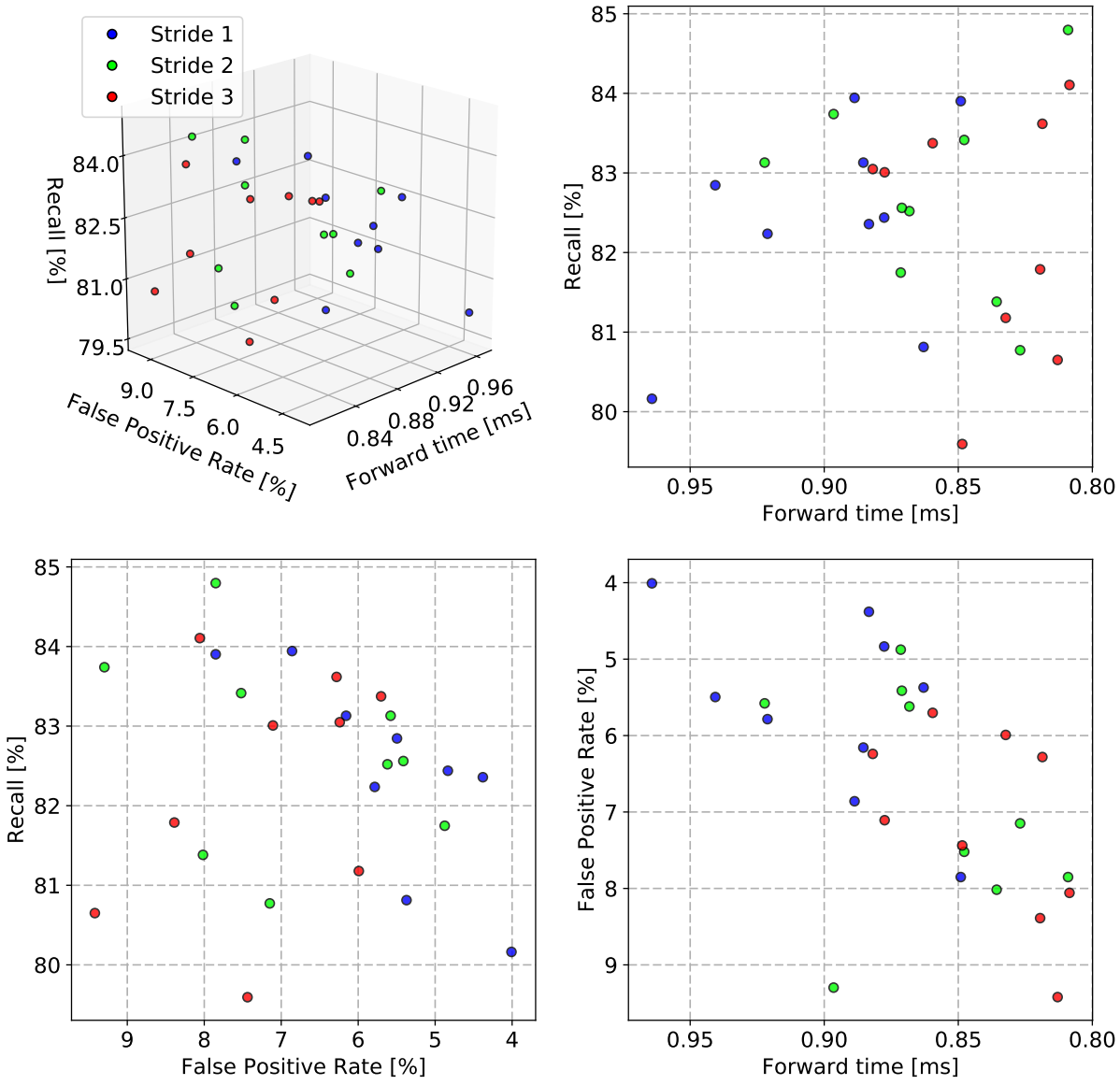
A.14 Cross-validation results of CNN Structure Analysis



A.15 List of all Networks for CNN Structure Analysis

Structure	Act.	Kernels	Size	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Score
CFF	ReLU	10	3	50	18x28	4.09	80.57	0.98	49.3833
CFF	ReLU	20	3	50	18x28	3.97	81.18	1.04	49.1067
CFF	ReLU	20	5	50	18x28	4.38	81.75	1.01	47.4367
CAMF	ReLU	20	7	0	18x28	4.67	83.21	1.01	46.1833
CMFF	ReLU	10	3	50	18x28	4.42	86.26	1.14	46.1
CFF	ReLU	10	5	50	18x28	4.92	82.52	0.98	45.0533
CAMFF	ReLU	10	3	50	18x28	3.8	87.03	1.42	44.4767
CAFF	ReLU	10	3	50	18x28	4.59	87.52	1.22	43.9
CMFF	ReLU	10	7	50	18x28	4.75	85.08	1.14	43.7267
CMFF	ReLU	20	7	50	18x28	4.75	86.5	1.2	43
CAFF	ReLU	20	3	50	18x28	4.42	88.05	1.35	42.4967
CAFF	ReLU	10	7	50	18x28	4.96	86.63	1.17	42.3833
CFF	ReLU	1	3	50	18x28	5.7	82.97	0.89	42.3233
CMFF	ReLU	20	5	50	18x28	4.88	86.71	1.2	42.29
CAMFF	ReLU	10	5	50	18x28	4.46	88.13	1.38	41.6833
CAMFF	ReLU	20	7	50	18x28	4.59	89.19	1.46	39.6567
CAFF	ReLU	20	5	50	18x28	5.12	87.93	1.29	39.4567
CAFF	ReLU	20	7	50	18x28	5.17	86.95	1.27	39.23
CMFF	ReLU	10	5	50	18x28	5.87	88.37	1.12	38.5033
CFF	ReLU	20	7	50	18x28	5.99	82.93	1	38.37
CAF	ReLU	20	7	0	18x28	6.61	83.29	0.83	38.17
CAMF	ReLU	20	5	0	18x28	5.99	84.19	1.05	37.79
CAFF	ReLU	10	5	50	18x28	5.54	86.87	1.23	37.7833
CAMFF	ReLU	20	5	50	18x28	4.96	90.45	1.47	37.6567
CAMF	ReLU	10	7	0	18x28	6.28	81.42	0.94	37.3267
CMFF	ReLU	20	3	50	18x28	5.74	87.56	1.21	37.2133
CAF	ReLU	20	5	0	18x28	6.94	84.39	0.86	35.9567
CAF	ReLU	20	3	0	18x28	6.86	84.67	0.89	35.93
CAMFF	ReLU	10	7	50	18x28	5.58	88.46	1.35	35.6733
CFF	ReLU	10	7	50	18x28	6.69	84.07	0.96	35.35
CAF	ReLU	10	5	0	18x28	7.6	83.01	0.77	33.3367
CAMF	ReLU	20	3	0	18x28	6.57	82.07	1.09	32.8033
CAMF	ReLU	10	3	0	18x28	7.02	82.36	0.96	32.8
CAFF	ReLU	1	3	50	18x28	6.78	85.2	1.12	31.9867
CMFF	ReLU	1	3	50	18x28	7.23	82.07	1.01	30.4433
CFF	ReLU	1	5	50	18x28	7.73	82.64	0.87	30.4333
CAMF	ReLU	10	5	0	18x28	7.98	83.98	0.95	27.78
CAF	ReLU	10	3	0	18x28	8.6	81.95	0.78	26.7833
CMFF	ReLU	1	5	50	18x28	8.06	82.8	1	25.9067
CAFF	ReLU	1	7	50	18x28	7.81	82.48	1.1	25.3
CAMFF	ReLU	1	3	50	18x28	8.06	87.2	1.22	22.9733
CAMFF	ReLU	1	5	50	18x28	8.22	84.76	1.23	21
CFF	ReLU	1	7	50	18x28	9.67	84.63	0.88	19.2567
CAMFF	ReLU	20	3	50	18x28	4.42	88.54	1.55	0
CAF	ReLU	10	7	0	18x28	11.2	82.4	0.76	0
CAMF	ReLU	1	3	0	18x28	3.68	14.11	0.82	0
CF	ReLU	1	3	0	18x28	21.2	63.09	0.45	0
CF	ReLU	10	3	0	18x28	19.59	65.53	0.51	0
CF	ReLU	20	3	0	18x28	20.17	67.28	0.57	0
CF	ReLU	1	5	0	18x28	19.67	61.1	0.44	0
CF	ReLU	10	5	0	18x28	18.6	63.82	0.51	0
CF	ReLU	20	5	0	18x28	20.25	66.46	0.56	0
CF	ReLU	1	7	0	18x28	20.37	62.03	0.45	0
CF	ReLU	10	7	0	18x28	19.83	65.73	0.5	0
CF	ReLU	20	7	0	18x28	17.77	62.97	0.54	0
CMFF	ReLU	1	7	50	18x28	10.17	84.96	1.01	0
CMF	ReLU	1	3	0	18x28	19.5	69.55	0.62	0
CMF	ReLU	20	3	0	18x28	13.1	83.78	0.78	0
CMF	ReLU	1	5	0	18x28	16.65	69.15	0.6	0
CMF	ReLU	10	5	0	18x28	13.06	82.15	0.7	0
CMF	ReLU	20	5	0	18x28	13.55	84.96	0.77	0
CMF	ReLU	1	7	0	18x28	20.79	72.8	0.61	0
CMF	ReLU	10	7	0	18x28	12.15	80.33	0.68	0
CMF	ReLU	20	7	0	18x28	12.07	83.54	0.72	0
CAFF	ReLU	1	5	50	18x28	5.25	68.13	1.1	0
CAF	ReLU	1	3	0	18x28	14.71	54.31	0.68	0
CAF	ReLU	1	5	0	18x28	17.98	50.57	0.65	0
CAF	ReLU	1	7	0	18x28	21.57	65.85	0.68	0
CAMFF	ReLU	1	7	50	18x28	6.36	68.54	1.23	0
CAMF	ReLU	1	5	0	18x28	13.35	40.41	0.84	0
CAMF	ReLU	1	7	0	18x28	12.36	39.23	0.82	0
CMF	ReLU	10	3	0	18x28	13.47	81.3	0.7	0

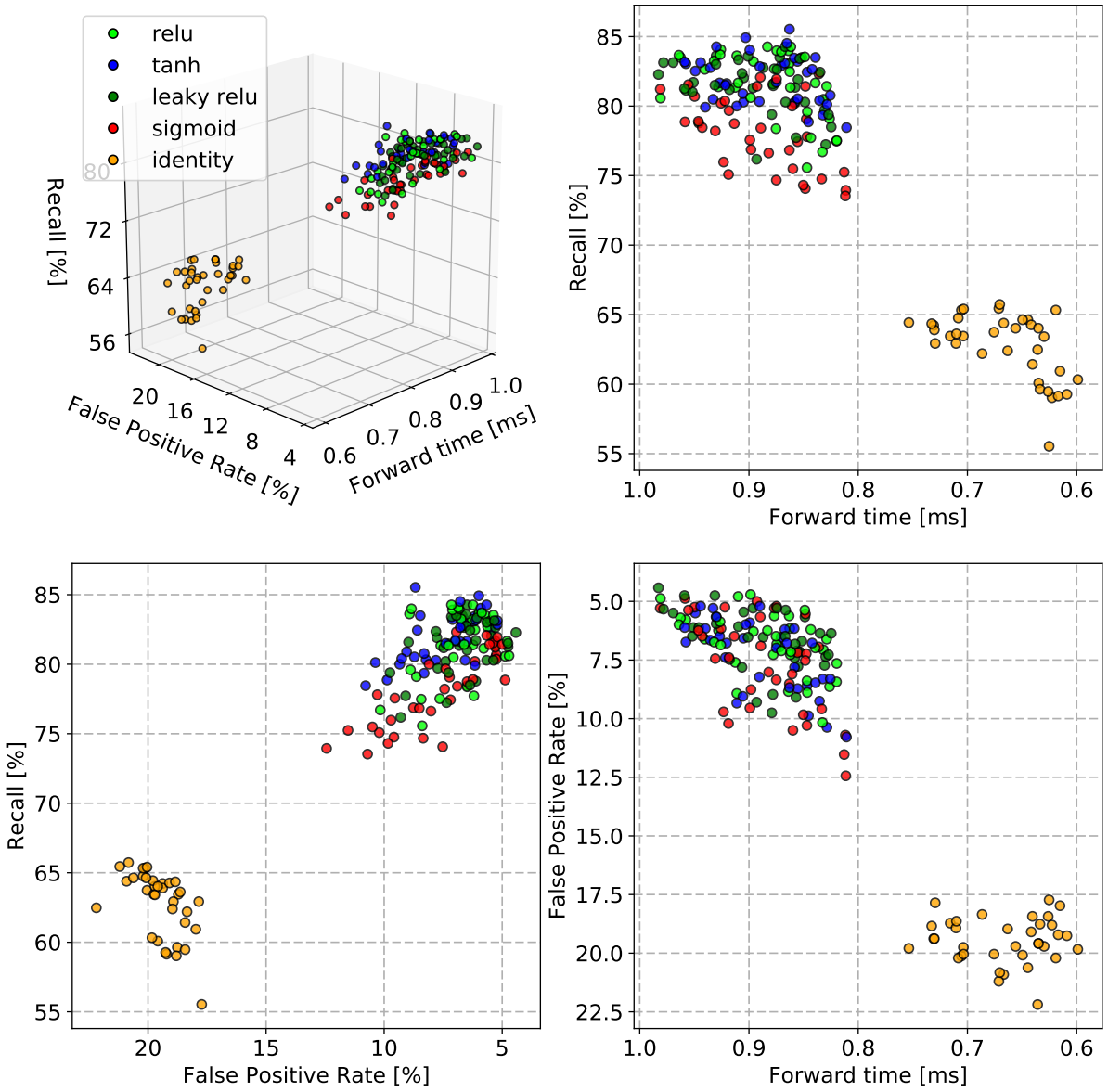
A.16 Cross-validation results of CNN Stride Analysis



A.17 List of all Networks for CNN Stride Analysis

Stride	Kernels	Size	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Score
1	2	3	50	18x28	4.38	82.36	0.88	50.24
1	5	3	50	18x28	4.01	80.16	0.96	50.1267
1	2	5	50	18x28	4.83	82.44	0.88	47.5667
2	5	3	50	18x28	4.88	81.75	0.87	47.2367
2	5	5	50	18x28	5.41	82.56	0.87	44.3267
1	1	5	50	18x28	5.37	80.81	0.86	44.1833
3	5	7	50	18x28	5.7	83.37	0.86	43.0567
2	5	7	50	18x28	5.62	82.52	0.87	43.0533
2	2	3	50	18x28	5.58	83.13	0.92	42.4967
1	5	5	50	18x28	5.5	82.85	0.94	42.4833
3	5	5	50	18x28	5.99	81.18	0.83	41.1867
1	5	7	50	18x28	5.79	82.24	0.92	40.94
3	2	3	50	18x28	6.28	83.62	0.82	40.46
1	2	7	50	18x28	6.16	83.13	0.89	39.6167
3	5	3	50	18x28	6.24	83.05	0.88	39.31
1	1	3	50	18x28	6.86	83.94	0.89	35.6867
2	2	5	50	18x28	7.15	80.77	0.83	34.09
3	2	7	50	18x28	7.11	83.01	0.88	34.0767
2	2	7	50	18x28	7.52	83.41	0.85	32.35
2	1	3	50	18x28	7.85	84.8	0.81	31.6333
3	2	5	50	18x28	7.44	79.59	0.85	31.5567
1	1	7	50	18x28	7.85	83.9	0.85	30.5333
3	1	3	50	18x28	8.06	84.11	0.81	30.1433
2	1	5	50	18x28	8.02	81.38	0.84	28.8733
3	1	5	50	18x28	8.39	81.79	0.82	27.19
3	1	7	50	18x28	9.42	80.65	0.81	20.83
2	1	7	50	18x28	9.3	83.74	0.9	20.78

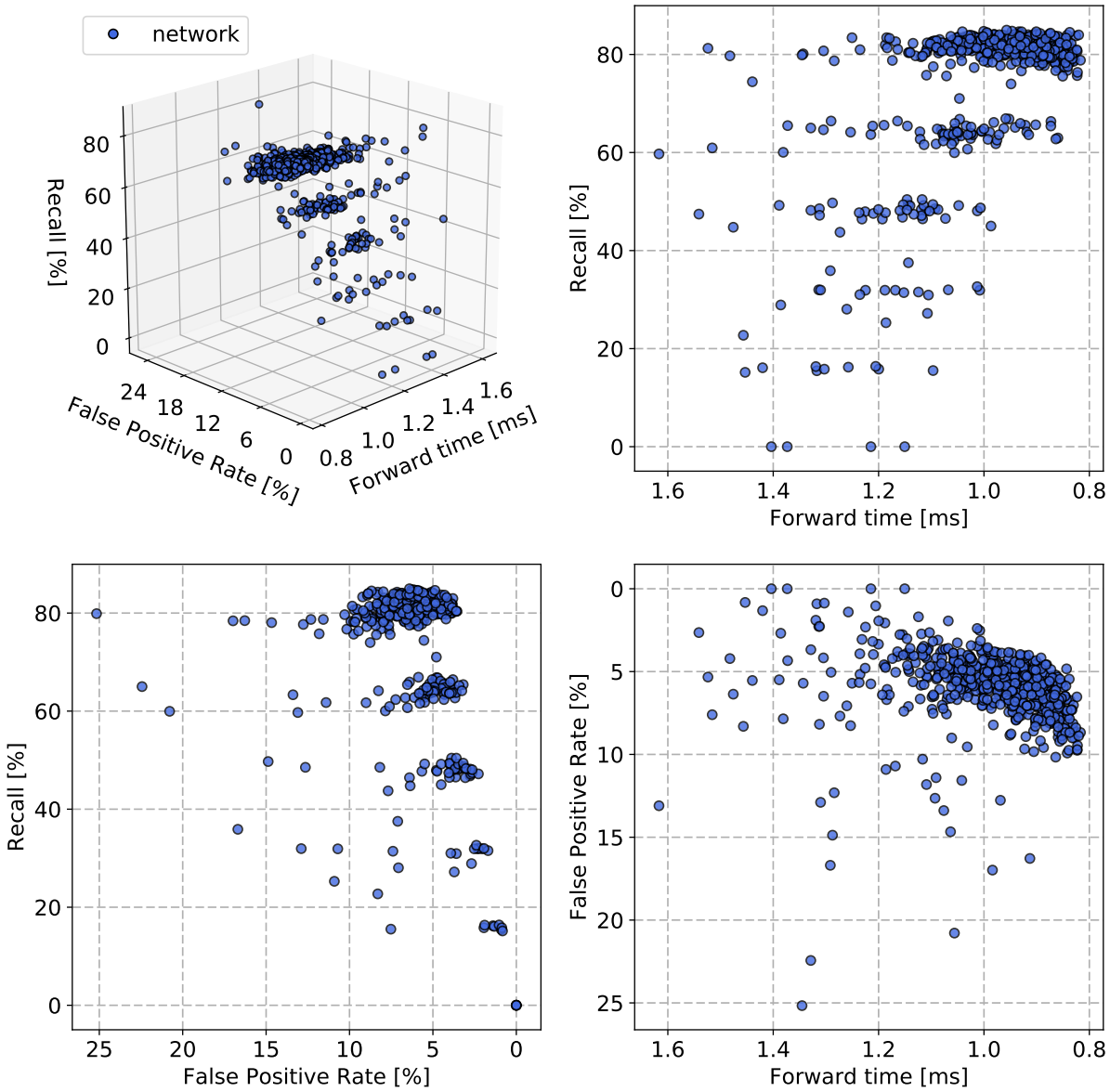
A.18 Cross-validation results of CNN Activation Function Analysis



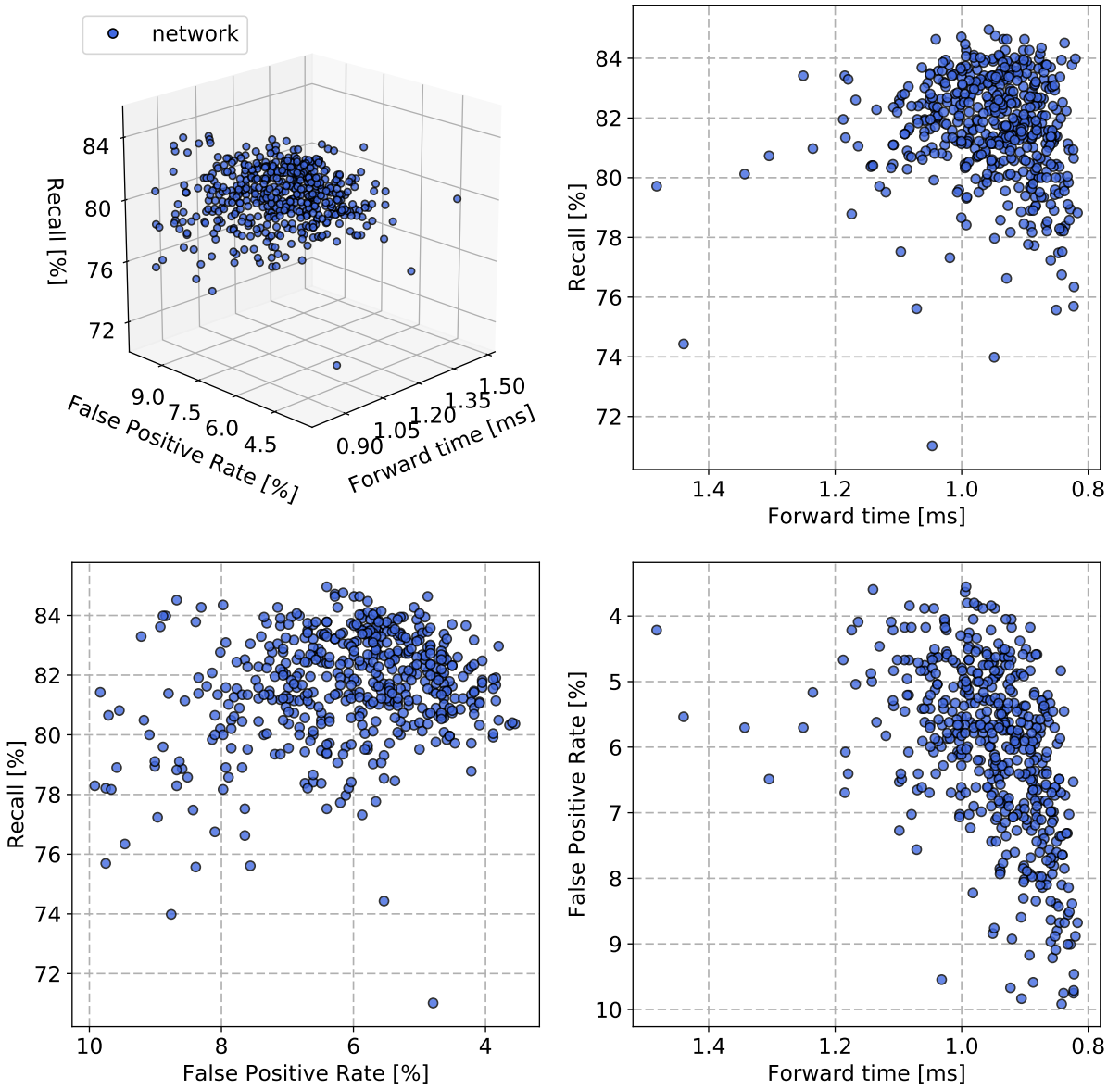
A.19 List of the 40 highest scoring Networks for CNN Activation Function Analysis

Activation	Kernels	Size	Neurons	Input	FPR[%]	Recall[%]	Avg. time[ms]	Score
ReLU	10	3	50	17x27	4.88	80.57	0.98	44.64
Sigmoid	2	5	50	17x27	5.29	81.67	0.88	44.55
Sigmoid	10	3	50	17x27	4.88	78.86	0.96	44.47
Leaky ReLU	2	5	50	17x27	5.29	81.3	0.9	44.02
Sigmoid	5	7	50	17x27	5.25	80.37	0.92	43.55
ReLU	1	3	50	17x27	5.62	83.05	0.86	43.43
Leaky ReLU	5	3	20	17x27	5.45	82.03	0.9	43.31
TanH	10	3	20	17x27	5.29	80.49	0.93	43.15
Sigmoid	5	5	50	17x27	5.25	80.69	0.95	43.06
ReLU	2	3	50	17x27	5.62	82.68	0.88	42.90
Leaky ReLU	2	3	20	17x27	5.66	80.28	0.85	42.46
Sigmoid	10	7	50	17x27	5.37	81.54	0.96	42.42
Sigmoid	10	5	50	17x27	5.29	81.22	0.98	42.4
Sigmoid	2	7	50	17x27	5.66	82.07	0.89	42.26
TanH	10	3	50	17x27	5.5	82.52	0.95	42.17
Leaky ReLU	10	7	50	17x27	5.5	83.13	0.97	41.97
Leaky ReLU	5	7	50	17x27	5.66	83.01	0.93	41.77
ReLU	10	5	50	17x27	5.7	83.5	0.96	41.1
TanH	2	5	20	17x27	6.16	83.46	0.86	40.32
Leaky ReLU	5	5	50	17x27	5.95	83.62	0.93	40.24
ReLU	2	5	20	17x27	6.16	82.48	0.86	40
Leaky ReLU	5	7	20	17x27	6.07	83.33	0.91	39.82
ReLU	10	3	20	17x27	5.79	81.14	0.96	39.77
TanH	1	3	50	17x27	6.2	83.09	0.87	39.76
ReLU	10	7	50	17x27	5.99	83.66	0.96	39.41
ReLU	5	5	20	17x27	6.28	83.21	0.9	38.72
TanH	10	7	50	17x27	6.12	83.17	0.96	38.47
Leaky ReLU	10	5	20	17x27	6.12	81.71	0.95	38.18
Sigmoid	5	3	50	17x27	6.16	80.2	0.93	37.84
ReLU	10	3	10	17x27	6.24	81.38	0.93	37.75
TanH	2	3	20	17x27	6.53	82.89	0.87	37.71
TanH	10	3	10	17x27	6.16	79.92	0.94	37.54
Leaky ReLU	1	3	50	17x27	6.69	83.5	0.85	37.36
Leaky ReLU	5	5	20	17x27	6.53	83.58	0.9	37.34
Leaky ReLU	2	7	50	17x27	6.53	82.76	0.89	37.27
Leaky ReLU	10	3	10	17x27	6.12	78.78	0.95	37.20
Sigmoid	10	5	20	17x27	6.24	78.9	0.95	36.52
TanH	5	3	20	17x27	6.49	81.67	0.93	36.35
ReLU	2	7	50	17x27	6.82	83.98	0.87	36.34
ReLU	5	3	10	17x27	6.49	78.37	0.88	36.25

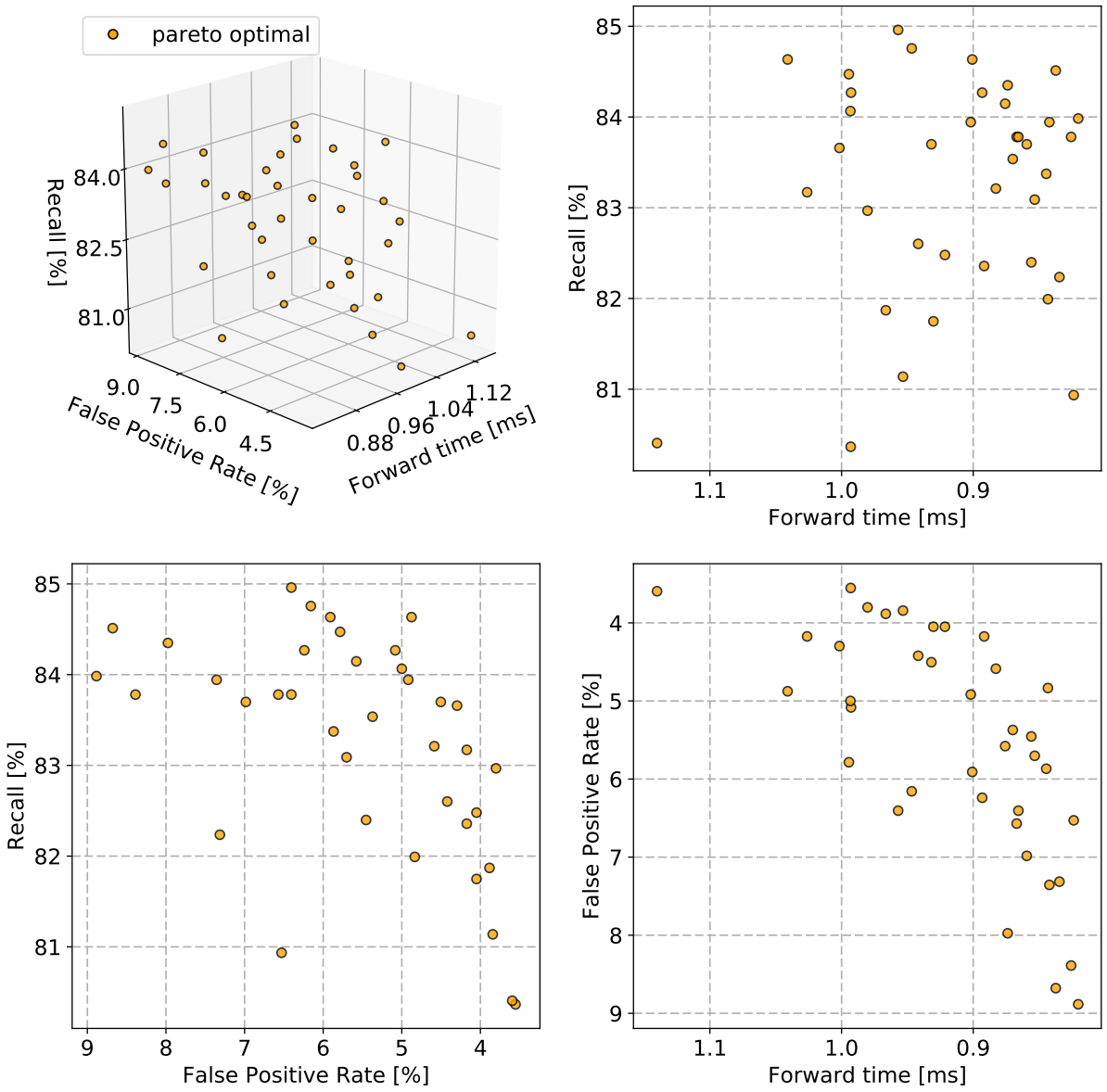
A.20 Faulty cross-validation results of all CNNs for analyzing Kernel Count, Kernel Size, Neuron Count and Input Dimensions



A.21 Cross-validation results of all CNNs for analyzing Kernel Count, Kernel Size, Neuron Count and Input Dimensions



A.22 Cross-validation results of pareto-optimal CNNs for analyzing Kernel Count, Kernel Size, Neuron Count and Input Dimensions



A.23 List of CNNs cut due to faulty training

Structure	Activation	Kernels	Size	Neurons	Input
CFF	ReLU	50	5	20	15x24
CFF	ReLU	50	3	50	18x28
CFF	ReLU	50	5	50	18x28
CFF	ReLU	50	7	50	20x31
CFF	ReLU	50	3	50	21x33
CFF	ReLU	50	5	20	21x33
CFF	ReLU	50	5	50	22x35
CFF	ReLU	50	5	50	23x36
CFF	ReLU	50	5	50	24x38
CFF	ReLU	50	3	10	14x22
CFF	ReLU	50	3	20	14x22
CFF	ReLU	50	5	10	14x22
CFF	ReLU	50	3	10	15x24
CFF	ReLU	50	3	20	15x24
CFF	ReLU	50	5	10	15x24
CFF	ReLU	50	7	10	15x24
CFF	ReLU	50	3	10	16x25
CFF	ReLU	50	3	20	16x25
CFF	ReLU	50	5	10	16x25
CFF	ReLU	50	5	20	16x25
CFF	ReLU	50	7	10	16x25
CFF	ReLU	50	3	10	17x27
CFF	ReLU	50	3	20	17x27
CFF	ReLU	50	3	50	17x27
CFF	ReLU	50	5	10	17x27
CFF	ReLU	50	7	10	17x27
CFF	ReLU	50	5	20	17x27
CFF	ReLU	50	3	10	18x28
CFF	ReLU	50	3	20	18x28
CFF	ReLU	50	5	10	18x28
CFF	ReLU	50	5	20	18x28
CFF	ReLU	50	7	10	18x28
CFF	ReLU	50	3	10	19x30
CFF	ReLU	50	3	20	19x30
CFF	ReLU	50	5	10	19x30
CFF	ReLU	50	5	20	19x30
CFF	ReLU	50	5	50	19x30
CFF	ReLU	50	7	10	19x30
CFF	ReLU	50	7	20	19x30
CFF	ReLU	50	3	10	20x31
CFF	ReLU	50	3	20	20x31
CFF	ReLU	50	3	50	20x31
CFF	ReLU	50	5	10	20x31
CFF	ReLU	50	5	20	20x31
CFF	ReLU	50	7	10	20x31
CFF	ReLU	50	7	20	20x31
CFF	ReLU	50	3	10	21x33
CFF	ReLU	50	3	20	21x33
CFF	ReLU	50	5	10	21x33
CFF	ReLU	50	7	10	21x33
CFF	ReLU	50	7	20	21x33
CFF	ReLU	50	3	10	22x35
CFF	ReLU	50	3	20	22x35
CFF	ReLU	50	5	10	22x35
CFF	ReLU	50	5	20	22x35
CFF	ReLU	50	7	10	22x35
CFF	ReLU	50	7	20	22x35
CFF	ReLU	50	7	50	22x35
CFF	ReLU	50	3	10	23x36
CFF	ReLU	50	3	20	23x36
CFF	ReLU	50	5	10	23x36
CFF	ReLU	50	5	20	23x36
CFF	ReLU	50	7	10	23x36
CFF	ReLU	50	7	20	23x36
CFF	ReLU	50	5	10	24x38
CFF	ReLU	50	5	20	24x38
CFF	ReLU	50	7	10	24x38
CFF	ReLU	50	7	20	24x38
CFF	ReLU	50	7	50	24x38
CFF	ReLU	50	3	10	24x38
CFF	ReLU	50	3	20	24x38
CFF	ReLU	50	3	50	24x38
CFF	ReLU	50	5	10	25x39
CFF	ReLU	50	5	20	25x39

Structure	Activation	Kernels	Size	Neurons	Input
CFF	ReLU	50	5	50	25x39
CFF	ReLU	50	7	10	25x39
CFF	ReLU	50	7	20	25x39
CFF	ReLU	50	7	50	25x39
CFF	ReLU	50	3	10	25x39
CFF	ReLU	50	3	20	25x39
CFF	ReLU	50	3	50	25x39
CFF	ReLU	20	5	20	14x22
CFF	ReLU	20	3	10	16x25
CFF	ReLU	20	7	50	17x27
CFF	ReLU	20	5	20	20x31
CFF	ReLU	20	3	10	15x24
CFF	ReLU	20	5	10	17x27
CFF	ReLU	20	3	10	17x27
CFF	ReLU	20	3	10	18x28
CFF	ReLU	20	7	10	18x28
CFF	ReLU	20	7	20	18x28
CFF	ReLU	20	3	10	19x30
CFF	ReLU	20	3	20	19x30
CFF	ReLU	20	3	10	20x31
CFF	ReLU	20	3	20	20x31
CFF	ReLU	20	7	10	20x31
CFF	ReLU	20	3	10	21x33
CFF	ReLU	20	3	10	22x35
CFF	ReLU	20	5	10	22x35
CFF	ReLU	20	7	10	22x35
CFF	ReLU	20	7	20	22x35
CFF	ReLU	20	3	10	23x36
CFF	ReLU	20	3	20	23x36
CFF	ReLU	20	5	10	23x36
CFF	ReLU	20	7	10	23x36
CFF	ReLU	20	7	20	23x36
CFF	ReLU	20	7	50	23x36
CFF	ReLU	20	5	10	24x38
CFF	ReLU	20	5	20	24x38
CFF	ReLU	20	7	10	24x38
CFF	ReLU	20	7	20	24x38
CFF	ReLU	20	3	10	24x38
CFF	ReLU	20	5	10	25x39
CFF	ReLU	20	7	10	25x39
CFF	ReLU	20	7	20	25x39
CFF	ReLU	20	3	10	25x39
CFF	ReLU	10	5	10	14x22
CFF	ReLU	10	5	10	19x30
CFF	ReLU	10	7	10	22x35
CFF	ReLU	10	3	20	23x36
CFF	ReLU	10	7	10	24x38
CFF	ReLU	10	5	10	25x39
CFF	ReLU	10	3	10	18x28
CFF	ReLU	10	7	20	19x30
CFF	ReLU	10	7	10	20x31
CFF	ReLU	10	3	10	21x33
CFF	ReLU	10	5	10	21x33
CFF	ReLU	10	7	10	21x33
CFF	ReLU	10	5	10	23x36
CFF	ReLU	10	5	10	24x38
CFF	ReLU	10	3	10	24x38
CFF	ReLU	10	3	20	24x38
CFF	ReLU	10	7	10	25x39
CFF	ReLU	10	3	20	25x39
CFF	ReLU	5	5	10	20x31
CFF	ReLU	5	5	10	22x35
CFF	ReLU	5	7	10	23x36
CFF	ReLU	5	5	10	24x38
CFF	ReLU	5	5	10	25x39
CFF	ReLU	2	5	10	22x35
CFF	ReLU	2	5	10	16x25
CFF	ReLU	2	5	10	18x28
CFF	ReLU	2	5	10	21x33
CFF	ReLU	2	7	10	21x33
CFF	ReLU	1	7	50	23x36
CFF	ReLU	1	7	10	17x27
CFF	ReLU	1	3	10	20x31
CFF	ReLU	1	5	10	21x33

A.24 List of pareto-optimal CNNs for analyzing Kernel Count, Kernel Size, Neuron Count and Input Dimensions

Kernels	Size	Neurons	Input	FPR	Recall	Avg. time	Max. Time	Score
5	3	50	23x36	3.55	80.37	0.99	1.66	52.35
10	5	50	16x25	3.8	82.97	0.98	1.63	51.92
2	3	50	24x38	3.84	81.14	0.95	1.6	51.67
5	3	50	16x25	4.05	82.48	0.92	1.57	51.46
2	3	50	18x28	4.17	82.36	0.89	1.29	51.3
10	3	50	18x28	3.88	81.87	0.97	1.33	51.27
5	3	50	18x28	4.05	81.75	0.93	1.62	51.01
2	3	50	19x30	4.59	83.21	0.88	1.34	49.26
10	3	50	25x39	3.6	80.41	1.14	1.74	49.07
5	5	50	19x30	4.5	83.7	0.93	1.61	48.96
5	3	20	18x28	4.42	82.6	0.94	1.53	48.88
5	5	50	24x38	4.17	83.17	1.03	1.33	48.77
5	5	50	22x35	4.3	83.66	1	1.48	48.75
1	3	20	20x31	4.83	81.99	0.84	1.36	48.21
5	5	50	14x22	4.92	83.94	0.9	1.29	47.12
2	3	20	17x27	5.37	83.54	0.87	1.28	44.89
20	5	50	18x28	5	84.07	0.99	1.63	44.89
10	5	50	22x35	4.88	84.63	1.04	1.53	44.79
10	7	50	16x25	5.08	84.27	0.99	1.66	44.47
2	5	20	19x30	5.45	82.4	0.86	1.27	44.23
2	7	50	18x28	5.58	84.15	0.88	1.37	43.63
1	3	50	14x22	5.7	83.09	0.85	1.47	43.16
1	5	50	16x25	5.87	83.37	0.84	1.24	42.43
2	7	50	20x31	5.91	84.63	0.9	1.52	41.41
10	7	50	18x28	5.79	84.47	0.99	1.63	40.28
2	5	50	17x27	6.24	84.27	0.89	1.46	39.51
10	7	50	15x24	6.16	84.76	0.95	1.58	38.96
1	5	50	19x30	6.4	83.78	0.87	1.4	38.79
1	3	10	19x30	6.53	80.93	0.82	1.31	38.06
2	7	50	16x25	6.57	83.78	0.87	1.18	37.77
2	7	50	22x35	6.4	84.96	0.96	1.44	37.38
1	3	50	15x24	6.98	83.7	0.86	1.26	35.48
1	5	50	14x22	7.36	83.94	0.84	1.3	33.68
1	5	50	15x24	7.31	82.24	0.83	1.53	33.62
1	7	50	19x30	7.98	84.35	0.87	1.61	29.50
1	5	20	14x22	8.39	83.78	0.83	1.32	27.65
1	7	50	15x24	8.68	84.51	0.84	1.37	25.95
1	7	20	19x30	8.88	83.98	0.82	1.32	24.98