University of Amsterdam
Faculty of Science
The Netherlands


# Detecting a checkered black and white football

6 EC Bachelor honoursproject

*Authors:*
Caitlin Lagrand
Douwe van der Wal
Pieter Kronemeijer

*Supervisor:*
Arnoud Visser

February 6, 2017

# Contents


**1 Introduction** **1**

**2 Data set** **2**

**3 Method** **2**
3.1 TensorFlow . . . . . 3
3.2 Pipeline . . . . . 4
3.2.1 Candidates . . . . . 4
3.2.2 Classifier . . . . . 5

**4 Results** **6**

**5 Discussion** **10**
5.1 Possible follow-up studies . . . . . 12

**6 Conclusion** **12**

**7 Installation guides** **12**
7.1 TensorFlow on DAS4 . . . . . 12
7.2 TensorBox . . . . . 13
7.3 OpenCV . . . . . 13
7.4 OpenCV on DAS4 . . . . . 14

**8 Appendix** **16**
8.1 A . . . . . 16

**Abstract**

This paper presents two ways of detecting a checkered black and white football that is used during soccer matches between robots in the SPL[1]. A well-considered trade-off between computation time and accuracy is of the utmost importance, since the ball detector has to be able to maintain a reasonable framerate while running on a Nao robot. The first method, a computationally heavy method using TensorFlow, shows an accuracy of 97.2% on images with a resolution of 1280x960. However, it takes 0.215 seconds to detect a ball on a laptop with a GPU and thus will not be fast enough to work on a Nao robot. The second, computationally light, method is able to detect a ball with an average computation time of 0.110 seconds on a Nao robot. However, the accuracy of this method is only 62.6%. Furthermore, 0.110 seconds is still a fraction too long. Soon, research will be conducted on ways to optimize this method in order to make it fast enough for robot soccer.

[1]Standard Platform League, only Nao robots without hardware modifications are allowed.

Figure 1: Nao robots during a match at RoHow[2](Hamburg).

# 1 Introduction

The Dutch Nao Team [6] plays autonomous robot soccer matches [4] in the Standard Platform League of the RoboCup. Playing soccer consists of many different challenges, such as movement, communication, vision and processing all the sensor data real-time. The idea behind the RoboCup is to be able to win against the 2050 FIFA world champion in a fair way. Until 2008 robot soccer was played with small dog-like robots and a bright orange ball[14]. From 2008 onward it is being played with humanoid robots (Nao's) on artificial grass (Figure 1). Last year, the color of the ball changed from the easily-detectable orange to a much harder detectable black-and-white pattern. Furthermore, this year the lighting conditions will change to window lighting where possible, which results in uneven lighting. This influences the color of the ball and a simple color-based method will not work well. In this paper several ways will be discussed to recognize this new ball while playing football. Both computationally expensive and inexpensive methods will be looked at. While a computationally inexpensive method would be preferred, since all computations have to be done on the robot itself during a match, computationally heavy methods can serve as a good comparison of the trade-off between speed and precision that is being made.

In the following sections of this paper, the approach that was taken to detect the ball will be explained and the results and conclusions will be presented. Section 3 will explain the different methods that were tried. In section 4 the results will be evaluated and Section 5 will discuss these results. Finally, in section 6, the conclusion will be presented.

[2]`https://rohow.de/2016/en/`


Figure 2: Different kind of variations within the data set.

## 2 Data set

The data set that was used, consists of 1526 images. These images are annotated manually by selecting a box around the ball. The images were taken at two different locations, in the IRL[3][12] at Science Park Amsterdam, and the TUHH[4]. At both locations multiple batches of images were taken, each with its own lighting condition. This varies from dark to bright, from smooth to lots of shadows and from artificial light to natural light. There is also a great variation as to the distance to the ball, whether the ball is moving or not and whether there are other objects in the picture, like robots or feet.

Figure 2 shows some of the different kinds of variation within the data set. The images have been recorded at a resolution of 1280*960 pixels. In order to get the most out of the trade-off between speed and process-ability, these images have been resized to resolutions of 640*480 pixels, 320*240 pixels and 160*120 pixels.

For the final comparison between the two chosen methods, a new testing data set has been made, consisting of 500 images in 2 lighting conditions made in 1 location (IRL).

## 3 Method

This section will go into detail about two approaches that were taken to detect the ball. Section 3.1 will explain a computationally expensive method using a Deep Neural Net. In section 3.2, a pipeline that presents a computationally inexpensive method will be considered. Other methods that were tried, will be shortly discussed in section 5.

[4]Intelligent Robotics Lab
[4]Technische Universität Hamburg (https://www.tuhh.de/tuhh/startseite.html)

### 3.1 TensorFlow

A machine learning ball detector which is not limited to the Nao robot's hardware could make be a good proof of concept of why the robots need more processing power. Besides that, a more low-end solution could perhaps be deduced from the high-end implementation.

Stewart, Andriluka, and Ng [9] describe a machine learning approach of recognizing people's heads on surveillance camera footage with Caffe [3]. Stewart, has also made an implementation using a TensorFlow[1] model, which is the model used in this section. The first part is GoogLeNet[10], which is pretrained on ImageNet and outputs features. Then a grid consisting of squares of 32*32 is put over the image and each box in the grid is processed seperately. If something has been found, the location is returned as a bounding box. The idea is to use this approach to recognize balls instead of heads.

A model has been trained for every resolution. These models have been trained for several hundred thousand iterations on a Nvidia TitanX, controlled by the DAS4 supercomputer. Training does not require a TitanX. Any device with TensorFlow installed can train this model, but it will usually take significantly longer.

The images in this data set must comply with certain requirements, namely that both the image width and height need to be divisible by 32, and both the hight and width need to be at least 224 pixels. A solution for using smaller images is putting a bar on the side or below the image in order to meet the specifications.

When a model has been properly trained a laptop can use it to process the video-feed from a Nao robot real time to inform the robot about the location of the ball.

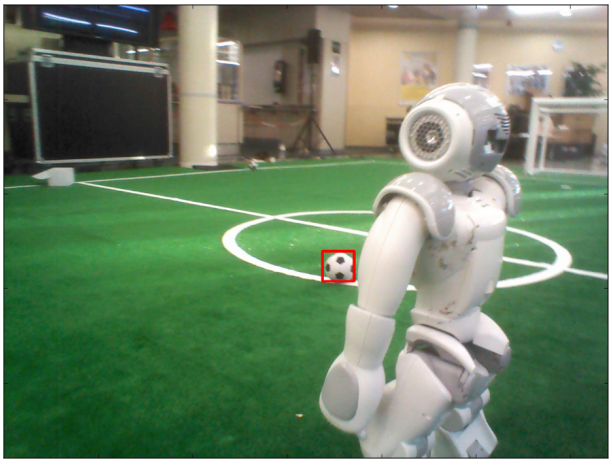


Figure 3: TensorBox output.

## 3.2 Pipeline

Since the ball detector needs to work real time on the Nao robot, performing a classification algorithm directly on the obtained images results in a computational heavy and slow ball detector. Therefore, the ball detector is split into two parts: obtaining candidates and a classification algorithm. Figure 4 shows the pipeline used to detect the ball. The pipeline uses an image as input and returns a x1,y1,x2,y2-box around the ball in that image.

First, the possible candidates are determined. Section 3.2.1 will explain how candidates are obtained. Next, the candidates are further explored using a classification algorithm. Section 3.2.2 will talk about the classifier that is used.

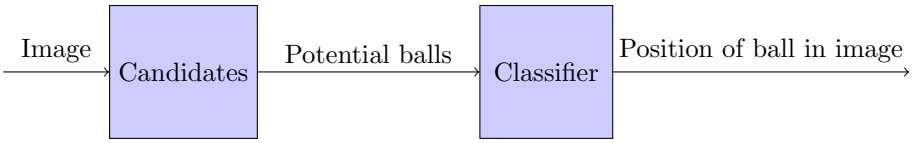



Figure 4: The pipeline of detecting a ball.

### 3.2.1 Candidates

Candidates are parts of the image that can be the ball. The ball has a checkered black and white pattern, thus parts of the image that contain this pattern could be the ball. Two methods to find candidates will be explained in this section. Other methods that were tried, will be shortly discussed in section 5.

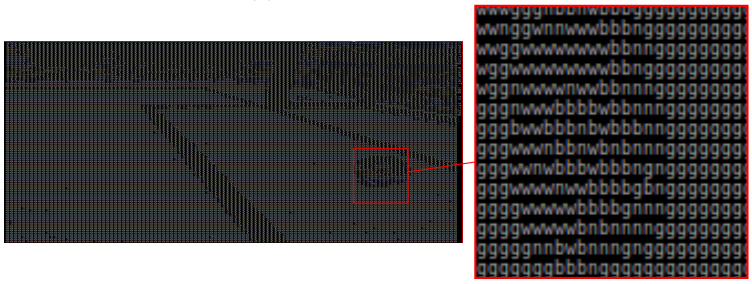
#### Regular expressions

The first method searches for black and white patterns using regular expressions [11]. Regular expressions are a fast way of finding pieces of text which match a certain description. To use regular expressions on images, each pixel in a column is converted to a character based on the color of that pixel: b(lack), w(hite), g(reen), n(one) [5]. Figure 5 shows the result of converting an image taken by the robot (5a) to strings (5b).

$$(b + .0, 5|w + .0, 5)+ \tag{1}$$

The color of a pixel is obtained from pre-defined color ranges. A regular expression (1) is applied to the resulting string to match patterns of a ball. This regular expression search for patterns of black or white pixels with some noise. A lot of matches are found that only represents a vertical pattern (column-wise). The green lines in Figure 6 represent these vertical patterns. The center of the match (black dot) is now used as location of a possible ball. This location is used to expand the possible ball horizontally with black or white pixels (red line). The horizontal and vertical patterns combined result in a box around a possible ball. This part of the image plus an added offset is now considered as potential ball and ready to be considered by the classifier.

(a) Image from the robot



(b) Image as strings

Figure 5: Converting an image to strings of b, w, g and n characters.

### Using the old position of the ball

During a match in the Standard Platform League, the ball is often not moving. The probability that the ball is still at the same position as the previous position is thus high. So, the old position can be a good candidate. An offset is added to this old position and is then used as candidate. If the ball is not found in the candidate by the classifier, the offset is again added to the candidate. This new candidate is now explored by the classifier. This process repeats until a ball is found or the whole image is used as candidate, see Figure 7.

### 3.2.2 Classifier

The classifier determines the position of the ball in an image. The Viola-Jones Classifier [13] from OpenCV [5] is used. This classifier is based on Haar-like features. The classifier is trained on two different kind of data sets. First of all, the data set explained in section 2 is used. The annotated parts of the images are used as positives for the classifier and the rest of the images is used as negatives. The other data set contains 10,887 ball candidates obtained using the regular expressions method explained in section 3.2.1. The candidates are manually annotated as ball or no ball. The images annotated as ball are added to the positives of the first data set and the images containing no ball are added to the negatives. The classifier is trained with images in the YCbCr color space

[5] http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

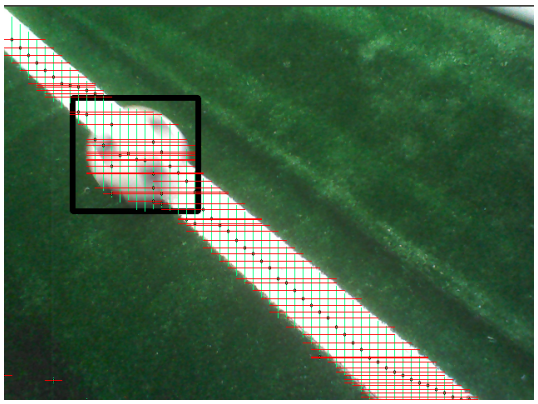
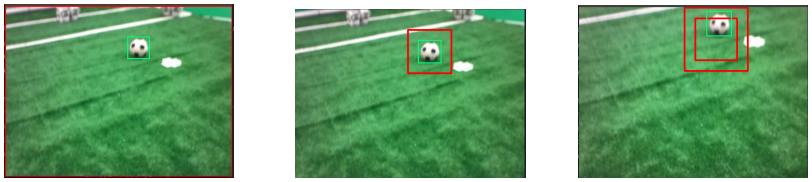


Figure 6: Candidates using regex. Black rectangle is the annotated ball, black dot is the center of the candidates, green line is the vertical regex pattern and red line is the horizontal expanding with black and white.



(a) First image, no previous ball.

(b) Next image, with previous ball.

(c) Next image, with previous ball increasing.

Figure 7: Using the old position as candidate.

since the framework used for playing football uses this color space as default and no conversion has to be performed before classifying the image. During the detection phase, a potential ball is evaluated by the classifier. If a ball is found within the potential ball, that part of the image is classified as ball and no other candidate is considered.

# 4 Results

Once the TensorFlow model has been trained, it is possible to annotate a 640*480 video feed at a reasonable frame rate of about 6 on an Nvidia 860M controlled by an i7 4710HQ. A sidenote on this is that a great deal of time is taken by getting an image from the Nao, when using locally saved images 15 images can be processed per second. Sadly, it is currently not yet possible to run the evaluation script on the Nao robot itself, for several reasons. Firstly, TensorFlow is not made for 32bit devices. This means that it is not possible to install TensorFlow on the Nao the regular way. Secondly, the Nao does not

have a GPU, which means that all the needed processing power will be taken of the CPU, which is already very busy running all the other tasks to play soccer. Therefore the amount of frames per second that can be processed by the Nao will drop significantly. This means that in the end using TensorFlow for balldetection will hurt the Nao's soccer playing ability more than it helps.

| **Resolution** | **Confidence** | **Accuracy (%)** | False Positives | **Time (s)** |
|---|---|---|---|---|
| 1280x960 | 0.3 | 97,2 | | 0.215 |
| 640x480 | 0.3 | 94,6 | | 0.059 |
| 320x240 | 0.3 | 93 | | 0.023 |
| 160x120 | 0.3 | 84.6 | 294 | 0.020 |
| | 0.6 | 83.6 | 261 | |
| | 0.9 | 76,8 | 185 | |

Table 1: TensorFlow accuracy per resolution.

| **Method** | **Offset** | **Accuracy (%)** | **False Positives** | **Time (s)** |
|---|---|---|---|---|
| Only Classifier | - | 76.75 | 30 | 0.0879 |
| Regex without strings | 30 | 40.6 | 4 | 0.0499 |
| Regex including strings | 30 | 40.6 | 4 | 0.386 |
| Regex without strings | 60 | 68.6 | 6 | 0.108 |
| Regex including strings | 60 | 68.6 | 6 | 0.427 |
| Regex without strings | 90 | 73 | 14 | 0.189 |
| Regex including strings | 90 | 73 | 14 | 0.494 |
| Old position | 30 | 80.8 | 20 | 0.257 |
| Old position | 60 | 80.4 | 17 | 0.167 |
| Old position | 90 | 80.8 | 19 | 0.132 |
| Old position + whole image | 30 | 76.6 | 19 | 0.0868 |
| Old position + whole image | 60 | 78 | 16 | 0.0822 |
| Old position + whole image | 90 | 78.2 | 19 | 0.0836 |

Table 2: Pipeline method with 640x480 resolution on a laptop.

The pipeline is evaluated with different kind of methods: only the classifier, the regular expressions method with or without converting to strings while measuring the time, and the old position method repeatedly growing or only one candidate followed by the whole image. Different offsets around a candidate are also taken into consideration per method. The result of performing the different methods on a laptop on images with a 640x480 resolution is shown in Table 2. Table 3 shows the accuracies of the different methods on images with a resolution of 320x240. Only the 320x240 resolution is used for testing with a robot. The times to detect a ball on a resolution of 320x240 are shown in Figure 4.

Because the old position method only works well when the data set is chronological, a special test data set was create to test this method: 174 images that

| Method | Offset | Accuracy (%) | False Positives |
|---|---|---|---|
| Only Classifier | - | 62.6 | 3 |
| Regex without strings | 30 | 43 | 1 |
| Regex including strings | 30 | 43 | 1 |
| Regex without strings | 60 | 57.2 | 1 |
| Regex including strings | 60 | 57.2 | 1 |
| Regex without strings | 90 | 56.6 | 1 |
| Regex including strings | 90 | 56.6 | 1 |
| Old position | 30 | 62.6 | 2 |
| Old position | 60 | 63.2 | 2 |
| Old position | 90 | 62.6 | 1 |
| Old position + whole image | 30 | 59.6 | 1 |
| Old position + whole image | 60 | 60 | 1 |
| Old position + whole image | 90 | 60.4 | 1 |

Table 3: Accuracy of the pipeline method with 320x240 resolution

represent a robot walking to a ball and kicking the ball away. The results of the algorithm are shown in Table 5 and Table 6. The times on the robot are recorded on a Nao version 4, which contains a ATOM Z530 1.6 GHz CPU.

| Method | Offset | Accuracy (%) | False Positives |
|---|---|---|---|
| Only Classifier | - | 88.5 | 1 |
| Old position | 30 | 90.2 | 0 |
| Old position | 60 | 90.2 | 0 |
| Old position | 90 | 90.2 | 0 |
| Old position + whole image | 30 | 90.2 | 0 |
| Old position + whole image | 60 | 90.8 | 0 |
| Old position + whole image | 90 | 90.8 | 0 |

Table 5: Accuracy of the pipeline method with 320x240 resolution on a chronological data set.

| **Method** | **Offset** | **Time (s) laptop** | **Time (s) robot** |
|---|---|---|---|
| Only Classifier | - | 0.0171 | 0.113 |
| Regex without strings | 30 | 0.0134 | 0.152 |
| Regex including strings | 30 | 0.0990 | 2.88 |
| Regex without strings | 60 | 0.03277 | 0.322 |
| Regex including strings | 60 | 0.121 | 3.05 |
| Regex without strings | 90 | 0.0398 | 0.457 |
| Regex including strings | 90 | 0.127 | 2.67 |
| Old position | 30 | 0.0450 (h: 0.196, l: 0.00166) | 0.455 (h: 1.64 l: 0.0158) |
| Old position | 60 | 0.0309 (h: 0.116, l: 0.00313) | 0.299 (h: 1.07, l: 0.0242) |
| Old position | 90 | 0.0249 (h: 0.0803, l: 0.00461) | 0.239 (h: 0.616, l: 0.0382) |
| Old position + whole image | 30 | 0.0177 (h: 0.0359, l: 0.00181) | 0.169 (h: 0.443, l: 0.0117) |
| Old position + whole image | 60 | 0.0195 (h: 0.0509, l: 0.00293) | 0.184 (h: 0.434, l: 0.0252) |
| Old position + whole image | 90 | 0.0200 (h: 0.0481, l: 0.00437) | 0.192 (h: 0.439, l: 0.0444) |

Table 4: Times of the pipeline method with 320x240 resolution.
h = highest time measured, l = lowest time measured

| **Method** | **Offset** | **Times (s) laptop** | **Time (s) robot** |
|---|---|---|---|
| Only Classifier | - | 0.0319 | 0.195 |
| Old position | 30 | 0.0448 (h: 0.147, l: 0.00373) | 0.312 (h: 1.23, l: 0.0224) |
| Old position | 60 | 0.0375 (h: 0.122, l: 0.00667) | 0.253 (h: 0.842, l: 0.0461) |
| Old position | 90 | 0.0433 (h: 0.113, l: 0.0122) | 0.249 (h: 0.699, l: 0.0699) |
| Old position + whole image | 30 | 0.0231 (h: 0.0834, l: 0.00376) | 0.110 (h: 0.374, l: 0.0230) |
| Old position + whole image | 60 | 0.0293 (h: 0.0793, l: 0.00776) | 0.139 (h: 0.309, l: 0.0444) |
| Old position + whole image | 90 | 0.0383 (h: 0.0919, l: 0.00922) | 0.175 (h: 0.364, l: 0.0692) |

Table 6: Times of the pipeline method with 320x240 resolution on a chronological data set.
h = highest time measured, l = lowest time measured

# 5 Discussion

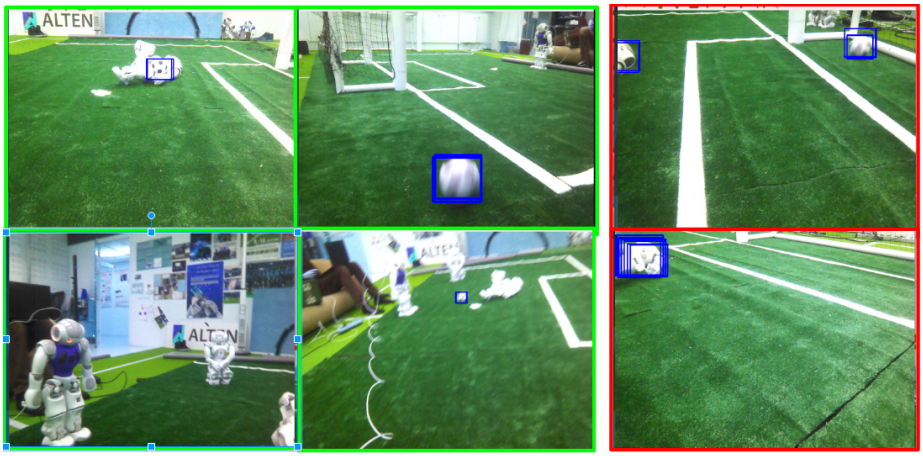


Figure 8: TensorBox results and mistakes.

Most of the mistakes made by TensorBox in the three highest resolutions are related to white objects near the border of an image. This is probably due to the fact that in the trainings set most balls on the edge of the image were rolling, and thus mainly white and hard to distinguish from a robot's head or other white objects. Perhaps these errors could be reduced when more data is added.

In the lowest resolution a lot of false positives occur at a low confidence. With a high confidence there are less false positives, but often balls are not seen either. The reason for the reasonable high accuracy with a high confidence is because there are less false positives on the images without a ball, and thus the algorithm classifies that image correctly.

The model, TensorBox, used in the machine learning approach worked very well, but balls are relatively easy to recognize compared to what the model was made for: heads on camera footage. Those heads can be either viewed from the back or front and have all kinds of different colors. For recognizing balls this model might be more than necessary and a less complex model could do just as good perhaps. It might be useful to try creating a less complex model and implement this in a machine learning library that does work on the Nao robots.

We also noticed that performance dropped and the amount of false positives became rather high at the 160*120 resolution. A reason for this could be that black bars were put around the 160*120 image from the Nao, instead of stretching out the image to the right resolution. The model processes 32*32 blocks, and by not stretching out the images the balls are a smaller part of a block, perhaps making it harder to recognize it.

Besides the complexity of the model, it can also only recognize one kind of item, so if one would want to recognize robots too, a second model would have

to be trained. This would become quite inefficient, running your image through more than one model to get all the information you need. To solve this problem a whole different model would be required, Redmon et al. [7] describes a model that can recognize up to twenty classes at a respectable speed.


Figure 9: Classifications

The accuracy of the pipelined method is much lower than the accuracy of the TensorBox. Most errors were balls near the border of the image or far away. Also, ball in the goal, were often a problem to detect. The false positives were mostly robot heads or feet. Figure 9 shows some results of the pipelined method.

As for the pipelined method, the results show that the old position method works better as candidate than the regular expression method. Also, converting images to strings, takes a long time and would be no option for a Nao robot with the current implementation. However, the Dutch Nao Team currently uses the regular expression method [5] and thus this method does already work fast enough on a Nao robot. An extension of this method with the classifier would be great to try. Nevertheless, the regular expression method still depends on color calibration. With different lighting circumstances, which are present during the RoboCup, as described in the introduction, a ball detector independent on calibration would be preferred.

Even on a non chronological data set, the old position method detects a ball within 0.2 seconds on a Nao robot. It still is two times too slow, but it is a good start. Next steps would be to optimize this algorithm to make it faster to be able to use it during matches.

Some other methods that have been tried involve the use of SIFT, SURF, ORB, and a simple blobdetector. All of these methods did not work properly in one way or another, which is why they were quickly abandoned. Detecting blobs did not work as good as expected, because the ball is not always circular. For example, when lying on a line, the ball is part of that line and no circle can

be found. SIFT, SURF and ORB turned out to be too slow to perform as good ball detector on the robot. For more information about these methods, see the appendix.

### 5.1 Possible follow-up studies

- Model capable of recognizing multiple objects instead of just one, for example the ball, other Nao robots, field lines and goals.
- Search for candidates without using any form of color calibration. This might be possible by looking at the saturation of a certain area.
- Neural networks working on Nao robots.

## 6 Conclusion

The TensorBox model has a vert high accuracy, but is currently not fast enough or even possible to work on a Nao robot. A pipelined method, using candidates and classify these candidates is a way of making a ball detector fast enough to run on a Nao. Because the ball is often not moving during a Standard Platform League match, the old position of the ball is a good candidate. Taking an offset of 30 around the old position followed by the whole image, if the ball was not found, works best. Running this algorithm on a Nao robot results in a 59.6% accuracy with on average 0.169 seconds to detect the ball when using a random data set.

## 7 Installation guides

### 7.1 TensorFlow on DAS4

A regular user does not have sudo rights on the DAS4, which makes installing things a bit harder than usual. A '.whl' file has been prepared on the das (/var/scratch2/koelma/tensorflow_pkg/cudaX.X-cudnn5.1). In the parent directory is a README file explaining how to install tensorflow. Don't forget to add the exports and added modules to the .bashrc file, otherwise TensorFlow will not work anymore after the terminal is closed.

This method will not work with the latest version of Anaconda (currently Anaconda 4.2.0)[3]. Anaconda version 2.0.1 has been tested and confirmed working. An installer for this Anaconda version can be found on the DAS4 in: /var/scratch2/koelma/src/Anaconda-2.0.1-Linux-x86_64.sh

In case an error related to having the wrong version of protobuf occurs, make sure that the version located at /usr/bin/protoc is used. It's possible to

[6] Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Continuum Analytics, Nov. 2016. Web. ¡https://continuum.io¿

check which version is being used by executing the command "which protoc" on the DAS. If the wrong version is being used, execute "$PATH = /usr/bin/protoc:$PATH". Keep in mind that the path will also need to be changed in the bashrc or this command will have to be executed every time TensorFlow is needed in a new terminal.

## 7.2 TensorBox

Follow the instructions on GitHub[8] to install TensorBox. An error related to 'Cython' might occur when running the example. To solve this problem, uninstall Cython using pip and then re-install it, again using pip.

In order to be able to run TensorBox with another data set than the example set, three things need to be changed.

1. A folder with a new data set needs to be added. The images in this data set must comply with certain requirements, namely that both the image width and height need to be divisible by 32, and both the hight and width need to be at least 224 pixels. A solution for using smaller images could be to put a bar on the side or below the image in order to meet the specifications.

2. Three json files with the image locations as keys and a list of the box(es) containing the objects that need to be detected as values. One of these three json files containing most of the data set should be used for training, another for testing and the third is for validating. The box should have the following format: [[x1, y1, x2, y2], [box2], [box3]].

3. The overfeat_rezoom.json contains variables influencing the training process that need to be set to the right values. The image size needs to be changed to the right size. The grid size is calculated by dividing the image size by 32, which should be a whole number. The location of the dictionary containing the image paths and boxes also needs to be set correctly.

When training the model will automatically save a model that can be used to make predictions after every 10.000 iterations. There will also be a file constantly growing in size and getting really large, you can throw this away after training.

## 7.3 OpenCV

OpenCV[2] can be installed through Anaconda using the command "conda install opencv". However, if you want to use OpenCV to process video's, this will not work. It is a know bug that the anaconda version of OpenCV will not work with video's[4]. Many solutions have been issued on the internet to fix this, but none seem to work properly. To be able to process video's with OpenCV, you

[7] https://github.com/ContinuumIO/anaconda-issues/issues/121
[8] https://github.com/jayrambhia/Install-OpenCV/tree/master/Ubuntu

will need to use a local version of Python (rather than the one from anaconda) and install OpenCV using an install script[5]. Manually building OpenCV from source is an option as a last resort, since this only worked for one out of three laptops.

### 7.4 OpenCV on DAS4

To install OpenCV on DAS4, one simply needs to add "/home/koelma/impala/third.14.11/x86_64-linux/lib/python2.6/site-packages" to the Pythonpath.

## References


[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[2] G. Bradski. In: *Dr. Dobb's Journal of Software Tools* ().

[3] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[4] Caitlin Lagrand et al. "Autonomous robot soccer matches". In: *BNAIC2016 Proceedings*. Nov. 11, 2016, pp. 237–238. URL: `http://bnaic2016.cs.vu.nl/images/bnaic/documents/BNAIC_2016_Proceedings.pdf`.

[5] Caitlin Lagrand et al. *Dutch Nao Team - Technical Report*. Tech. rep. Universiteit van Amsterdam, FNWI, Oct. 14, 2016. URL: `http://www.dutchnaoteam.nl/wp-content/uploads/2016/11/TechReport_DNT_2016.pdf`.

[6] Caitlin Lagrand et al. *Team Qualification Document for RoboCup 2017, Nagoya, Japan*. Tech. rep. Science Park 904, Amsterdam, The Netherlands: University of Amsterdam, Nov. 30, 2016. URL: `http://www.dutchnaoteam.nl/wp-content/uploads/2016/11/Team_Qualification_Document_2017.pdf`.

[7] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 779–788.

[8] S Russel. *TensorBox*. `https://github.com/TensorBox/TensorBox`. 2016.

[9] Russell Stewart, Mykhaylo Andriluka, and Andrew Y Ng. "End-to-end people detection in crowded scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2325–2333.

[10] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[11] Ken Thompson. "Programming Techniques: Regular Expression Search Algorithm". In: *Commun. ACM* 11.6 (June 1968), pp. 419–422. ISSN: 0001-0782. DOI: 10.1145/363347.363387. URL: http://doi.acm.org/10.1145/363347.363387.

[12] Camiel Verschoor, Patrick de Kok, and Arnoud Visser. *Intelligent Robotics Lab*. Tech. rep. Universiteit van Amsterdam, June 13, 2013. URL: http://staff.fnwi.uva.nl/a.visser/publications/Vision_Document.pdf. published.

[13] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–I.

[14] Arnoud Visser et al. *Dutch Aibo Team: Technical Report RoboCup 2006*. Tech. rep. Dutch Aibo Team, Dec. 1, 2006. URL: http://staff.fnwi.uva.nl/a.visser/publications/DAT2006TechReport.pdf. published.

# 8 Appendix

## 8.1 A

```
import sys
import numpy as np
import cv2
from naoqi import ALProxy
import time

import tensorflow as tf
import matplotlib.pyplot as plt
import os
import json
import subprocess
from scipy.misc import imread
import random

from train import build_forward
from utils import train_utils
from utils.annolist import AnnotationLib as al
from utils.stitch_wrapper import stitch_rects
from utils.train_utils import add_rectangles
from utils.rect import Rect
from utils.stitch_wrapper import stitch_rects
from evaluate import add_rectangles

def turnred(leds):
   leds.setIntensity('AllLedsRed', 1.0)
   leds.setIntensity("AllLedsBlue", 0.0)
   leds.setIntensity("AllLedsGreen", 0.0)

def turngreen(leds):
   leds.setIntensity('AllLedsRed', 0.0)
   leds.setIntensity("AllLedsBlue", 0.0)
   leds.setIntensity("AllLedsGreen", 1.0)


def StiffnessOn(proxy):
   # We use the "Body" name to signify the collection of all joints
   pNames = "Body"
   pStiffnessLists = 1.0
   pTimeLists = 1.0
   proxy.stiffnessInterpolation(pNames, pStiffnessLists, pTimeLists)

# NAO INIT
if(len(sys.argv) <= 1):
   print "parameter error"
   print "python " + sys.argv[0] + " <ipaddr> <port>"
```

```
   sys.exit()
ip_addr = sys.argv[1]
port_num = 9559# get NAOqi module proxy

# open needed proxies
videoDevice = ALProxy('ALVideoDevice', ip_addr, port_num)# subscribe top
    camera
leds = ALProxy("ALLeds",ip_addr,9559)
motionProxy = ALProxy("ALMotion", ip_addr, 9559)
postureProxy = ALProxy("ALRobotPosture", ip_addr, 9559)

# camera init
AL_kTopCamera = 0
AL_kVGA = 2
AL_kBGRColorSpace = 13
captureDevice = videoDevice.subscribeCamera("test", AL_kTopCamera,
    AL_kVGA, AL_kBGRColorSpace, 10)# create image
width = 640
height = 480
image = np.zeros((height, width, 3), np.uint8)



turnred(leds)
# Set NAO in Stiffness On
StiffnessOn(motionProxy)


# Tensorbox INIT
hypes_file = 'path/to/hypes/file.json'
iteration = INSERT NUMBER
with open(hypes_file, 'r') as f:
   H = json.load(f)

tf.reset_default_graph()
x_in = tf.placeholder(tf.float32, name='x_in', shape=[H['image_height'],
    H['image_width'], 3])
if H['use_rezoom']:
   pred_boxes, pred_logits, pred_confidences, pred_confs_deltas,
       pred_boxes_deltas = build_forward(H, tf.expand_dims(x_in, 0),
       'test', reuse=None)
   grid_area = H['grid_height'] * H['grid_width']
   pred_confidences =
       tf.reshape(tf.nn.softmax(tf.reshape(pred_confs_deltas, [grid_area
       * H['rnn_len'], 2])), [grid_area, H['rnn_len'], 2])
   if H['reregress']:
      pred_boxes = pred_boxes + pred_boxes_deltas
else:
   pred_boxes, pred_logits, pred_confidences = build_forward(H,
       tf.expand_dims(x_in, 0), 'test', reuse=None)
```

```
saver = tf.train.Saver()

# load model
framecounter = 0
start = time.time()

with tf.Session() as sess:
   sess.run(tf.initialize_all_variables())
   saver.restore(sess, 'path/to/folder/with/savefile/save.ckpt-%d' %
       iteration)
   kleurboollijst = [0,0, 0,0, 0,0, 0,0, 0,0]
   starttime = time.time()
   annolist = al.AnnoList()

   # main annotation loop
   while True:
     # get image
     if (time.time() - start) > 1:
        start = time.time()
        print framecounter
        framecounter = 0
     framecounter += 1
     result = videoDevice.getImageRemote(captureDevice);
     if result == None:
        print 'cannot capture.'
     elif result[6] == None:
        print 'no image data string.'
     else:
        # transform robotoutput to image
        values = map(ord, list(result[6]))
        img = np.reshape(values, (height,width,3)).astype('uint8')
        # check for balls using model
        feed = {x_in: img}
        (np_pred_boxes, np_pred_confidences) = sess.run([pred_boxes,
            pred_confidences], feed_dict=feed)
        # check if balls present
        bool = 0

        for elem, box in zip(np_pred_confidences, np_pred_boxes):
           elem2 = elem[0]
           if elem2[1]>0.6:
              bool = 1


        kleurboollijst.append(bool)
        kleurboollijst = kleurboollijst[1:]
        ntrue = sum(kleurboollijst)
        # change eyecolor if needed
        if ntrue > 7:
           turngreen(leds)
```

```
elif ntrue < 4:
   turnred(leds)
# draw box on image based on where the ball is
new_img, rects = add_rectangles(H, [img], np_pred_confidences,
    np_pred_boxes,
                      use_stitching=True, rnn_len=H['rnn_len'],
                          min_conf=0.3,
                      show_suppressed=True)


# update value if rectangle has found
if len(rects) > 0:
   averageX = (rects[0].x1 + rects[0].x2)/2
   averageY = (rects[0].y1 + rects[0].y2)/2
else:
   # set default value
   averageX = width/2
   averageY = height/2

# show image
cv2.imshow("Balldetection", new_img)

# press esc to exit
if cv2.waitKey(10) == 27:
   break

# make sure movement is not made too often
if (time.time() - starttime) > 0.1:
   starttime = time.time()
   # Yaw is turning , left is positive, right is negative.
   currentAngle = motionProxy.getAngles("HeadYaw", True)[0]
   # move head only when ball is surely spotted
   if ntrue > 5:
      # turning head
      if averageX < width/3:
         if currentAngle < 1.95:
            motionProxy.setAngles("HeadYaw", currentAngle + 0.06,
                0.6)
      if averageX > (width/3)*2:
         if currentAngle > -1.95:
            motionProxy.setAngles("HeadYaw", currentAngle - 0.06,
                0.6)
      # moving head up and down
      currentAngle = motionProxy.getAngles("HeadPitch", True)[0]
      if averageY > 380:
         if currentAngle < 0.51:
            motionProxy.setAngles("HeadPitch", currentAngle +
                0.09, 0.6)
      if averageY < 150:
         if currentAngle > -0.65:
```

```
                    motionProxy.setAngles("HeadPitch", currentAngle -
                        0.09, 0.6)

# shut down Nao
motionProxy.stiffnessInterpolation("Body", 0.0, 1)
motionProxy.stopMove()
print "Bye!"
```