20th RoboCup International Symposium, Leipzig, Germany, July 2016.

# Ball Localization for Robocup Soccer using Convolutional Neural Networks

Daniel Speck, Pablo Barros, Cornelius Weber, Stefan Wermter

University of Hamburg, Department of Informatics,
Knowledge Technology, WTM
Hamburg Bit-Bots,
Vogt-Koelln-Strasse 30, D - 22527 Hamburg, Germany
2speck@informatik.uni-hamburg.de

**Abstract.** In RoboCup soccer, ball localization is an important and challenging task, especially since the last change of the rule which allows 50% of the ball's surface to be of any color or pattern while the rest must remain white. Multi-color balls have changing color histograms and patterns in dependence of the current orientation and movement. This paper presents a neural approach using a convolutional neural network (CNN) to localize the ball in various scenes. CNNs were used in several image recognition tasks, particularly because of their capability to learn invariances in images. In this work we use CNNs to locate a ball by training two output layers, representing the x- and y-coordinates, with normal distributions fitted around the ball. Therefore the network not only locates the ball's position but also provides an estimation of the noise. The architecture processes the whole image in full size, no sliding-window approach is used.

**Keywords:** robocup, convolutional neural network, deep learning, tensorflow, ball detection, ball localization, noise, filtering

## 1 Introduction

In RoboCup humanoid soccer standard computer vision algorithms frequently utilize color and edge information for ball tracking [2, 6, 8, 17], because such algorithms are rather easy to implement and do not require lots of test data. For example, one of the common solutions is to search for round shapes in the picture and try to find the center of this shape [2]. Most of the standard algorithms are computationally cheap and deliver usable results. However, since there is no intelligent decision whether an object is a ball or not, they detect false positives quite often. Furthermore, since the complexity of the tasks of RoboCup has increased [20], the motivation for new solutions is growing. For instance, until 2014 the ball's color was all orange, but from 2015 onwards the specifications have changed to a ball with at least 50% white color leaving the rest of the ball open for any color combinations [20]. These changes in complexity adversely affected the results of many algorithms which were used up to now, because the color distribution now significantly changes in dependency of the camera's direction (plus lighting) and the orientation of the ball itself. Figure 1 shows the contrasting appearance of the ball for different orientations and movement.

The new ball specifications motivate intelligent approaches that are less dependent on assumptions like a homogeneous color of the ball. Moreover, technical advances make neural approaches possible in terms of computational power [1]. In sum the hypothesis of this paper is that a neural architecture should outperform standard algorithms which were used so far and especially be able to learn invariants for getting a much better rate of true positives and reduce misclassification. In general we think that without preprocessing and with a sufficiently large dataset a deep neural architecture should be able to learn the mentioned invariants, because the raw information covers not only more variation but also a more distinctive feature set. LeCun et. al. have already shown that deep networks are able to learn visual features for comparable tasks in robotics [15]. Here, we present a CNN which localizes the ball and outputs a distribution that can be utilized for determining the noise in the input signal.

CNNs have often and successfully been used for object classification tasks [7, 12, 14, 22] while in the last years also several CNNs for object localization have been proposed and showed good results [3,4,16,19,21]. However, e.g. sliding-window approaches are not optimal for ball localization in RoboCup soccer. Due to the large change in size of the features representing the ball when it is moving away or towards a robot the ball would cover several segments of a sliding-window solution quite often. Therefore we decided to let our architecture always classify the full image and manipulate the output, not the input, by feeding a probability distribution over the width and height of the image as the teaching signal. Apart from that, deep learning architectures like convolutional neural networks seem to be novel in the RoboCup humanoid league for object localization tasks.

For evaluating our architectures we randomly chose single frames of a non-moving ball or a few frames out of sequences as test images and measured the accuracy. In the sequences the ball is moving in various directions. We started with clean scenes with just a few objects and the ball, but our current dataset contains more images with robots, goal posts and miscellaneous objects (tables, chairs, windows, doors, humans, heaters) in the background.

In the second section we go into detail about our proposed architecture. In section 3, we describe the methodology of our approach: the used data set, the experiments in detail, our evaluation method and results. Finally, in section 4, we present our conclusion and possible future work.

## 2 Proposed Architecture

### 2.1 Convolutional Neural Networks

Fully-connected neural networks (e.g. MLPs) use lots of memory when they are designed for complex computer vision tasks. The vast amount of weights

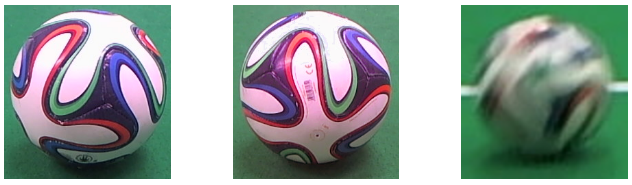

**Fig. 1.** One of the balls currently used in Robocup. The white color of the ball is almost identical compared to the color of the goal posts and the endlines. In the center image one can see the overemphasized red channel; this happens occasionally with our robot's camera and the lighting in our laboratory. The left and center images show non-moving balls with a distance of 0.5 meters to the robot. The right image shows a moving ball with a distance of 1.0 meter to the robot.

increases rapidly with the network's size. In contrast, CNNs filter the input information in the first layers [12]. The 2D data of an image is convolved by different kernel filters which extract characterizing features in the convolution layer. In subsequent layers this information is pooled and subsampled [12]. At the end different filters extracted the features by being applied over an image. These can be color distributions, shapes, specific patterns and so on. Properly designed and trained deep architectures can supply robust, state-of-the-art results [24]. In our approach we utilize the strengths of CNNs in classification tasks and combine this with altering the output layer to predict the ball's location with a distribution instead of absolute coordinates or bounding boxes.

### 2.2 Teaching Signal

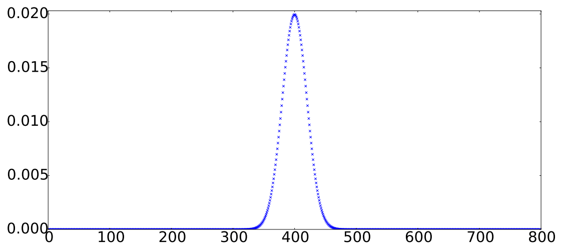

**Fig. 2.** A sample normal distribution that is taken as the teaching signal for the x output layer of the network. In this hypothetical sample image the ball is located at $x = 400$, therefore the normal distribution has a $\mu$ of 400. The width $\sigma$ is 20.

Our teaching signal is a normal distribution fitted around the center of the ball's real coordinates. Hence, one 800-dimensional vector for the x output layer (width of image) and one 600-dimensional vector for the y output layer (height of image). The normal distributions parameter $\mu$ is the center of the ball, while $\sigma$

was set to 20. An example of one teaching signal is shown in Figure 2. Using a normal distribution as the teaching signal ("labels") lets the network converge faster since a pixel-precise prediction would be harder to learn and has no substantial advantage in ball localization or tracking. Furthermore, the shape of the network's output data can be interpreted for approximating the current process's noise: in clean images the distributions (output) mostly had maxima of similar width, height, and shape. Reflections, blurring, and other visual distortions frequently caused wider maxima and noised shapes. For that reason the use of probability distributions as the teaching signal was a huge benefit, enabling a faster convergence and supplying additional information about the noise in the input data. Moreover Larochelle et al. stated that a high number of solutions that deliver a small training error increase the chance of the network converging into local minima [13]. In consequence, there is a decreased chance for the network to learn invariants and represent a generalization of the problem, ensuring the test error to be low, too. By using a probability distribution it is far less likely for the network to find a solution that only results in a decreased training error, because a maximum in the distribution consists of many classes being activated, therefore there are far less possible solutions with a low training error rate. Figure 2 shows an example for a probability distribution fed to the network for training.

### 2.3 Activation Functions

We used three different activation functions: the "traditional" rectified linear units (ReLU) [18], rectified linear units with an upper bound of 6 (ReLU6) [11], and soft-sign activation [5]. A ReLU activation is defined by:

$$h(x) = max(0, x) \ . \tag{1}$$

where $x$ represents a feature vector (the input for one complete layer) and $h$ is the activation transformed by the activation function, calculating the output of one layer. The networks using ReLU6 converged faster regularly. ReLU6 activation is defined by:

$$h(x) = min(max(0, x), 6) \ . \tag{2}$$

The upper bound of 6 in ReLU6 activation preserves the network of having too high activations since every activation runs into a hard limit and saturation. Additionally the precision of floating point numbers is higher around 0 and gets lower with values much higher than 0. As a consequence, our network learned features faster with ReLU6. A faster convergence and slightly better test results were provided by using soft-sign activation:

$$h(x) = \frac{x}{|x| + 1} \ . \tag{3}$$

The soft-sign activation function's shape is comparable to $tanh(x)$ but it saturates slower. This makes the soft-sign activation less dependent on weight initialization, which improves the forward activation as well as the backward learning [5].

### 2.4 Weight Initialization

One of the most challenging parts for deep neural networks is the initialization of weights [5]. If the weights are initialized to high the signal diverges while being propagated through the network; if the weights are too small it converges to zero or falls into a local minimum. The first scenario results in an activation of the output layer which does *not* deliver usable results. Instead of a proper distribution with an activity bump, the activation is large for all neurons while the second scenario results in an output that is very similar for nearly every input signal – the delta between different outputs for different inputs is very small. Many networks showed a convergence when their weights were initialized with a normal distribution $\mathcal{N}(\mu, \sigma^2)$, where $\mu = 0$ and $0.01 <= \sigma^2 <= 0.2$. This was the case throughout many different parameter settings in the majority (about 60%) of experiments. However, the results of the first experiments depended on heavy empirical testing which was very time-consuming. Better results were delivered by *normalized initialization* [5] (sometimes also referred to as *Xavier initialization*) which gives a good approximation for initializing the weights. This is especially useful because this approach showed good results in combination with soft-sign activation. Basically, the weights $\boldsymbol{W}$ between layer $j$ and layer $j+1$ are initialized with a uniform distribution (U) with upper and lower bounds defined by:

$$\boldsymbol{W} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] . \qquad (4)$$

where $n_j$ is the amount of neurons in layer $j$ and $n_{j+1}$ the amount of neurons in layer $j+1$. Besides the distribution shown in equation 4 it is also possible to approximate the variance of a normal distribution with normalized initialization by taking equation 4 as the variance for a normal distribution and set $\mu = 0$. For ReLU activation this initialization also works but has to be scaled up a bit because of the ReLU activation's lower bound of 0.

### 2.5 Optimization Algorithms

The experiments started with "traditional" gradient descent optimization for the learning algorithm but big data sets need a huge amount of training steps for the networks to converge. This problem was solved by using a stochastic gradient descent algorithm: Adam. Adam was developed by Kingma et. al. and successfully proposed especially for deep networks with a high amount of parameters [10].

### 2.6 Models

For demonstrating that networks with less neurons actually can achieve at least comparable results in ball localization we decided to develop two architectures. Model 1 delivered the best results for our experiments so far and after achieving this accuracy we abstracted from model 1 to create a network with less neurons to lower the computational costs of running the architectures.

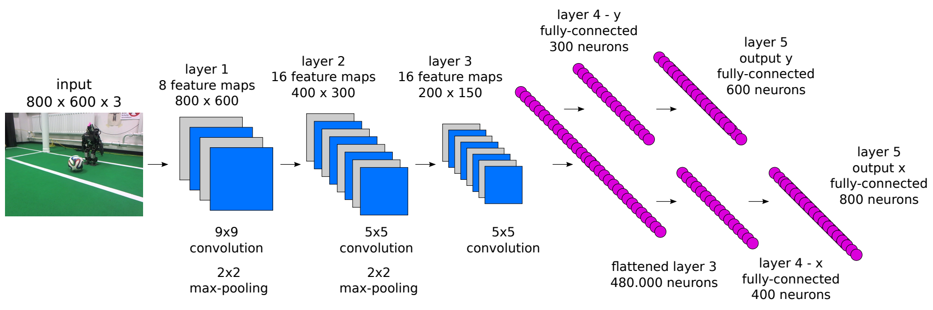

**Fig. 3.** Graphical illustration of the architecture of model 1.

**Model 1** is illustrated in Figure 3. It has only three convolutional layers, pooling is applied on the first two layers. The aim was to find out if less convolution and especially less pooling leaves more "raw" information that could be interpreted by the fully-connected classifiers in the last layers. The model was evaluated with and without dropout. If dropout was used, it was applied on every layer except the output layer with a dropout rate of 0.5 for training and no dropout for testing. Instead of dropping single connections, always whole neurons have been dropped out. The initial bias for the fully-connected layers was zero and 0.01 for the convolutional layers, which showed a marginally faster convergence for this architecture.
**Model 2**, illustrated in Figure 4, has a decreased number of fully-connected neurons. The training time for 10.000 training steps was reduced by 20% in comparison to model 1. Further decreasing the training time needs a more aggressive use of pooling to reduce the dimensionality, but in our case this procedure lowered the test error rates drastically. Again, dropout is applied on every layer but the output layer with the same rates for training and testing as model 1.

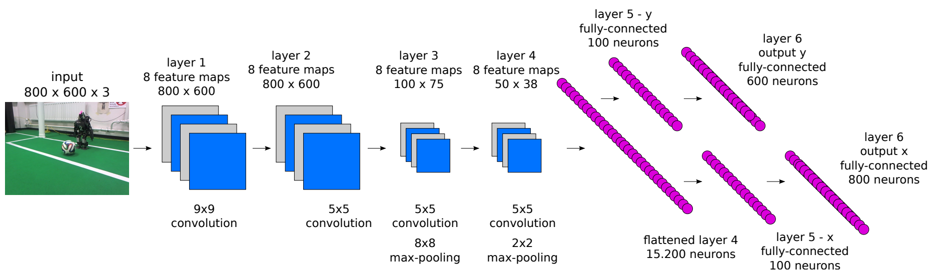

**Fig. 4.** Graphical illustration of architecture model 2.

## 3 Experimental Results

### 3.1 Dataset

CNNs need a large training data set [15]. All images of our data set have been recorded in our laboratory, which has a play field set up according to the RoboCup humanoid league rules but scaled down in size. The dataset contains 1.160 images; 80 are used as test images while 1.080 are used for training the network. The images portray various scenes, some are very clean with just the ball on the field, others cover a goalkeeper robot, the goal posts, a striker robot, and several arbitrary objects in the background (like tables, chairs, a heater, windows, a door). See Figure 5 for examples. Mostly the distance to the ball is between 0.5 and 5.0 meters. 400 images show a non-moving ball at various locations, while the rest are sequences of a moving ball. Roughly 20% of the training and test images contain a robot, a goal post or something similar right next to the ball and in almost 50 images the ball is partly covered, e.g. by the legs of a robot. The images are of dimension $800 \times 600 \times 3$ (width $\times$ height $\times$ RGB-channels) without *any* preprocessing. Hence, the dataset contains images with reflections, blurry images, images with overemphasized color channels, and so forth. Especially the red channel is intensified in some images, letting white walls appear pink.

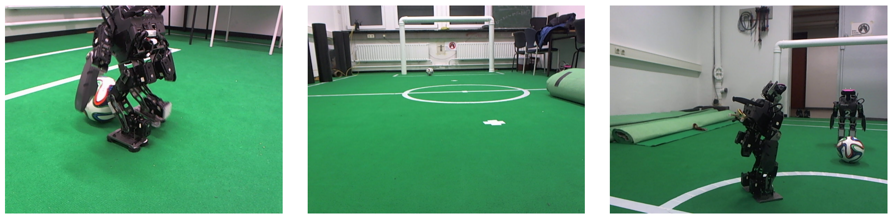
**Fig. 5.** Illustration of some test images used for evaluation, taken from the robot's camera.

### 3.2 Experimental Methodology

For evaluation we use a *top-11 error rate*, it describes if the top-11 activations (activity bump of the network) in one feed-forward step matches with the top-11 values of the teaching signal. Due to the symmetric shape of a normal distribution this guarantees the top-11 activations of the network to be 5 pixels around the real center of the ball. Hence, we basically count how many of the top-11 activations of the network are found in the teaching signal and build the mean over this. Only if the ball and therefore the normal distribution is exactly at a corner, this has a worst case of 10 pixels around the ball's center.

### 3.3 Results

Overall all networks showed a convergence after about 15.000 to 30.000 training steps. For the easier, cleaner images they started to converge after approximately 8.000 training steps, while about 15.000 training steps were needed for the more complicated ones. Table 1 shows the full results of our networks classifying the test images. Model 1 (soft-sign) offered the best results. It outperformed model 2 and the other activation functions. Regardless of which model was used, soft-sign activation delivered the best results, ReLU6 was always off by some percent and ReLU even lower. Hyperbolic tangent and sigmoid activation were only tested in the very first experiments and were dropped due to unsatisfactory results.
The network's output is visualized by plotting the top-11 prediction as well as a heatmap on top of the test images. The heatmaps and the distribution plots, illustrated in Figure 6, show that the network is classifying accurately.

**Table 1.** Results for full training (40.000 training steps).

| network | top11 x peak | top11 y peak |
|---|---|---|
| model 1 soft-sign | **81%** | **75%** |
| model 1 ReLU6 | 74% | 71% |
| model 1 ReLU | 72% | 69% |
| model 2 soft-sign | 71% | 70% |
| model 2 ReLU6 | 66% | 68% |
| model 2 ReLU | 65% | 63% |

### 3.4 Architecture Benchmarks

For an evaluation of our architecture's running time we used the full network (model 1) on a modern laptop (Intel Skylake i7 U-Series; mobile processor for low power consumption: 15W TDP). Running the complete test data set took this processor 74.43$s$, with a mean of 0.91$s$ per image. Even on our new Hambot robots the full architecture is too big for the RAM (2GB). Hence, we scaled down model 1 and retrained it. The architecture (layers, configuration, ...) is the same, but the input size is $200 \times 150$ and the output vectors for the x- and y-axis are 200- and 150-dimensional vectors. With this configuration our robots were able to load and work with the network. A processing of the full test data set took our robot 24.05s (laptop: 2.079s), with a mean of 0.304s (laptop: 0.026s) per image. The performance of model 1's downscaled version dropped: to roughly correspond to the top-11 error we evaluated a top-3 error which was 58% (top3 x) and 52% (top3 y) in total. Especially images with a ball-robot distance of over 2 meters dropped drastically in performance (some of these images were below 30%). Near distances up to 1 meter mostly showed error rates of roughly 70%. Thus, either faster processors for robots are necessary (to run the bigger nets) or low classification rates at medium to high distances have to be accepted.

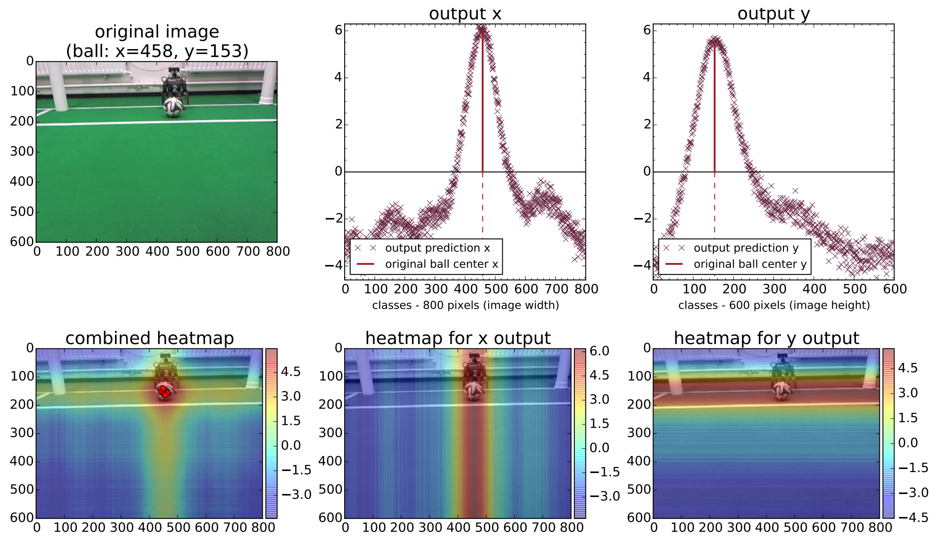

**Fig. 6.** In the first row the original test image is displayed. The real coordinates $(x, y)$ for the ball are $(458, 153)$. The x- and y-output, generated by the network for this specific test image, is also shown in row 1. The second row shows the heatmaps for the combined output (left), the x-output (center), and the y-output (right). The top11-activation is plotted into the left image in the second row with red dots.

## 4 Discussion

In most experiments the soft-sign activation networks dominated in performance, hence it is the best choice for this particular network design and the task of ball localization. Additionally the normalized initialization led to better results and faster convergence. Overall, model 1 as well as model 2 delivered usable results, while model 1 was the better choice, guaranteeing a very precise prediction in about 80% of the test images. For more complex images the discrepancy to the real coordinates often was less than ±10 pixels. Even results with a higher discrepancy may be useful in a game since the approximated location of the ball is much more important than a pixel-precise knowledge of it's position. Additionally, when catching the ball as a goal keeper or kicking the ball as a striker, pixel-precise information is unnecessary, since the hardware is not able to kick or catch balls with such a precision. As a consequence, distributions that point in the right direction are an efficient way for ball localization, especially when the discrepancy to the real coordinates is low enough. Furthermore this gives an approximated idea about the uncertainty in the input signal: clean shaped distributions cover low noise and high accuracy, while noisy input data distorts the distribution's shape. Despite this, less than 5% of the test images were classified incorrectly, leading to unusable results. In 90% of the test images the ball was found reliably and the rest showed noisy activity bumps. However, even these noisy activations were stable enough to filter the output, e.g. with a DBSCAN algorithm. Nonetheless, our architecture relies on a big training data set and the prediction results worsen a lot, if the input signal differs much from the training data. Thus, there is still work to be done. For example, our training data set is too small: in some situations the endlines or goal posts (because of their white color and in some angles round-shaped appearance) shift the network's attention, moving the activity bumps in between the ball and these objects. This most likely happens because many images cover no goal posts or only small parts of it. Also, scenes showing a partly covered ball led to mis-localizations, and scenes with no ball falsely led the network to output some location. Slightly less than half of such scenes were classified correctly. As a consequence, for such cases, there are clearly not enough training samples and therefore invariances learned by the network to produce reasonable output.

## 5 Conclusion and Future Work

We proposed a deep neural architecture, which is able to reliably locate the ball, although no pre-training was used. If supplied with enough training data (showing different ball orientations, speeds/trajectories, color distributions, distances, and other characteristics), it should outperform standard algorithms. The low computational power of robots compared to a computer limits the size of the network. For that reason we concentrate on developing an architecture for achieving satisfactory accuracy while keeping in mind the limited computational power of the robots. Hence, the current setup achieves the results with just a few layers and not *too* many neurons. The current architectures take no longer than 10 to 20 hours to train and can be run on modern robots with small changes. Our networks are able to deliver a reasonable performance in locating *multi-color* balls with arbitrary color patterns. Additionally, our architecture not only predicts the ball's location but does this with producing a distribution over the full width and height of the image. Therefore it delivers information about the noise, also. Figure 6 shows that for an accurate prediction the distributions on the output layer have a single bump with a spiky maximum. Additionally the x- and y-output bumps are of a similar shape (width, height). This can be seen in the majority of the test images when fed to the network, which proves the idea of the distribution output.

One aspect for developing a better architecture, to learn a better representation of invariances, will be to create a larger, more complex data set. Therefore more images are needed where the ball is partly covered, as well as images with no ball at all. Even mirroring the training data set's images improved our results: the top-11 accuracy of the test images went up to **83%** (top-11 x) and **76%** (top-11 y) for model 1. Thus, a bigger training data set with more varying images should stabilize the results. Up to now our data set only has few images that hide the ball e.g. between the legs of a robot. Learning inhibitory feedback should further improve the performance so images with no ball at all and a zero learning signal as well as more images with robots or other objects partially covering the ball should be the next milestone.

Another goal is to filter the noise and to predict the ball's movement over several frames with a recurrent neural network. As already mentioned, the cluster points even for test images with a low accuracy are close enough to each other to approximate the right direction to the ball. One possible and interesting solution is a neural Kalman filter [9]. It predicts a new/subsequent value for some observed feature based on previous input and observation noise as well as process noise. A neural implementation renders this task more complicated but also more dynamical, which should increase the filter's precision in the dynamical environment of RoboCup humanoid soccer. Standard implementations are used often but their success heavily relies on the knowledge of process and measurement errors. Szirtes et al. have shown that a neural architecture can learn predictable features along with the noise on specific features of the input information [23]. This information can be filtered by specific local neurons rendering the solution highly dynamical for feature rich, noisy environments. A way to keep the filter and prediction part computationally cheap and neurally reasonable is to use an Elman network. Potentially future work could connect both parts of the architecture even more: when the recurrent neural network outputs the next prediction of the ball's location, this information could be used to direct the attention of the convolutional neural network to the corresponding area in the image. Deep architectures can supply very good, robust results for those tasks [24].

## Acknowledgement

We would like to thank Stefan Heinrich for reviewing this paper, the Hamburg Bit-Bots[1] (esp. Nils Rokita, Fabian Fiedler), for assistance in working with the robots and giving feedback, and Nathan Lintz for constructive discussions about TensorFlow[2], which was used to build our architecture. The work was made in collaboration with the TRR 169 "Crossmodal Learning", funded by the DFG, and partially supported by CAPES Brazilian Federal Agency for the Support and Evaluation of Graduate Education (p.n.5951135).

## References

1. Bestmann, M., Reichardt, B., Wasserfall, F.: Hambot : An Open Source Robot for RoboCup Soccer (2015)
2. Coath, G., Musumeci, P.: Adaptive arc fitting for ball detection in robocup. Proceedings of APRS Workshop on Digital Image Analysing, Brisbane, Australia pp. 63–68 (2003)
3. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: Deep Neural Networks for Object Detection. Advances in Neural Information Processing Systems pp. 2553–2561 (2013)
4. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: Scalable Object Detection Using Deep Neural Networks. 2014 IEEE Conference on Computer Vision and Pattern Recognition pp. 2147–2154 (2014)

[1] `http://robocup.informatik.uni-hamburg.de/`
[2] `https://www.tensorflow.org/`

5. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. Aistats 9, 249–256 (2010)
6. Hanek, R., Schmitt, T., Buck, S., Beetz, M.: Towards robocup without color labeling. RoboCup 2002: Robot Soccer World Cup VI pp. 179–194 (2002)
7. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv: 1207.0580 pp. 1–18 (2012)
8. Jamzad, M., Sadjad, B.S., Mirrokni, V.S., Kazemi, M., Chitsaz, H., Heydarnoori, A., Hajiaghai, M.T., Chiniforooshan, E.: RoboCup 2001: Robot Soccer World Cup V, chap. A Fast Vis, pp. 71–80. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
9. Kalman, R.E.: A New Approach to Linear Filtering and Prediction Problems (1960)
10. Kingma, D.P., Ba, J.L.: Adam: a Method for Stochastic Optimization. International Conference on Learning Representations pp. 1–13 (2015)
11. Krizhevsky, A., Hinton, G.: Convolutional Deep Belief Networks on CIFAR-10. Unpublished manuscript (2010)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. Advances In Neural Information Processing Systems pp. 1–9 (2012)
13. Larochelle, H., Bengio, Y., Louradour, J., Lamblin, P.: Exploring Strategies for Training Deep Neural Networks. The Journal of Machine Learning Research 10, 1–40 (2009)
14. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient Based Learning Applied to Document Recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
15. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems pp. 253–256 (2010)
16. Malik, J., Girshick, R., Donahue, J., Darrell, T.: Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 580–587 (2014)
17. Murch, C., Chalup, S.: Combining Edge Detection and Colour Segmentation in the Four-Legged League. Australasian Conference on Robotics and Automation (ACRA2004) (2004)
18. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th International Conference on Machine Learning (ICML-10) pp. 807–814 (2010)
19. Oquab, M.: Is object localization for free? Weakly-supervised learning with convolutional neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition pp. 685–694 (2015)
20. RoboCup-Team: RoboCup Soccer Humanoid League Rules and Setup (2015)
21. Sermanet, P., Eigen, D., Zhang, X.: Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)
22. Szegedy, C., Reed, S., Sermanet, P., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions pp. 1–12 (2014)
23. Szirtes, G., Póczos, B., Lrincz, A.: Neural Kalman filter. Neurocomputing 65-66, 349–355 (2005)
24. Zhang, K., Liu, Q., Wu, Y., Yang, M.H.: Robust Visual Tracking via Convolutional Networks. CoRR abs/1501.0, 1–18 (2015)