Towards Real-Time Ball Localization using CNNs

Daniel Speck, Marc Bestmann, Pablo Barros

University of Hamburg, Department of Informatics,
Vogt-Koelln-Strasse 30, D - 22527 Hamburg, Germany
{2speck, bestmann, barros}@informatik.uni-hamburg.de

Abstract. Convolutional Neural Networks (CNNs) have shown promising results for various computer vision tasks. Despite their success, localizing the ball in real-world RoboCup Soccer scenes is still challenging. Especially considering real-time requirements and the limited computing power of humanoid robots. Another important reason is the lack of training and test data as well as baseline models to start with or compare to. In this paper, we propose a state-of-the-art ball detection model and make our training (over 35k images) and test (over 2k images) data sets publicly available.

Keywords: robocup · deep learning · dataset · ball detection · ball localization · fully convolutional neural network · tensorflow

1 Introduction

Ball localization is one of the essential skills in RoboCup Soccer. It has to be precise for close balls to allow the robot to position itself for example to shoot the ball, but it also has to be able to detect balls that are several meters away. The latter will become more difficult in 2020 when the playfield size will be doubled [6]. Additionally, it has to perform on the limited hardware of a humanoid robot in real-time, while still leaving resources for the other tasks of the robot.
Many approaches using neural networks were made since a change in the rules introduced multi-colored balls. Often classifiers are used to detect if a region of interest (ROI) contains a ball [8,10]. To the best of our knowledge, one of the first approaches working on full-scale raw images in RoboCup was proposed by us in RoboCup 2016, Leipzig [13]. It was trained on 1,080 training and 80 test images. The network's output consists of probability distributions that get combined to form a heatmap showing the likelihood of a pixel being part of a ball. While this showed promising results, the runtime performance of the network was too slow to be used on non-GPU ARM hardware robots during a game.
Schnekenburger et al. followed the same approach of taking the full image as input but used an FCCN [12]. This network was only trained on the center points of objects using 2,150 training and 250 test images. It was able to run in real-time, but the used robot has significantly more computational power. We present a model that is able to run on an NVIDIA Jetson TX2, a hardware that

is commonly used in the Humanoid Kid- and Teen-Size League. We train this architecture on 35,327 images and have 2,177 test images for evaluation.
We would like to contribute to the community and support the development of deep learning ball detection architectures. Therefore, we make our training and test datasets publicly available and also share our baseline architectures. This allows benchmarking and comparison of different approaches as well as an easy access to high-quality training data which is especially difficult for new teams.
The remainder of this paper is structured as follows: First, the data sets, as well as the metrics for measuring the detection quality are presented in section 2. Two models for locating balls are then presented in section 3. The results are afterward presented in section 4 and discussed in section 5.

2 Hamburg Bit-Bots Ball Dataset 2018

We propose the Hamburg Bit-Bots Ball Dataset 2018. All images and our models have been made public by us to encourage further scientific advances. The data can be accessed via our website[1]. The image sets can also be downloaded separately from our teams profile page on our Imagetagger[2] and the models are accessible at the corresponding GitHub repository[3]. We hope this supports the development and comparability of deep learning based models in RoboCup.

2.1 Data

The training dataset consists of 35,327 images (see Figure 1) and the test dataset of 2,177 images. Moreover, we supply an additional dataset with images only recorded on our robot for testing purposes that consists of another 764 images. We labeled these images with bounding boxes using the Hamburg Bit-Bots Imagetagger[4], an online tool we developed for making image annotation processes easier [3]. The training dataset is split into different so-called *image sets*. Over 14,000 images are from RoboCup 2016, Leipzig, Germany and nearly 8,000 from RoboCup 2017, Nagoya, Japan, over 6,000 images from our new lab, over 5,000 images from Iran Open 2018, and around 1,000 images from our old lab (without artificial turf) in Hamburg. Hence, the training images were recorded at six different locations. To boost the diversity we recorded different types of image sets: two different games from RoboCup 2017, many different sequences (us kicking or rolling the ball), non-moving balls at different angles and positions on the playfield as well as shots taken during preparation phases. Additionally, we have included another 14,886 negative samples, i.e. images covering the playfield, goals, a few robots, ..., but no ball, from RoboCup 2016, Leipzig, Germany, for evaluating models against false positives.

[1] `https://robocup.informatik.uni-hamburg.de/en/documents/bit-bots-ball-dataset-2018/`
[2] `https://imagetagger.bit-bots.de/users/team/1/`
[3] `https://github.com/Daniel451/Towards-Real-Time-Ball-Localization-using-CNNs`
[4] `https://imagetagger.bit-bots.de/`

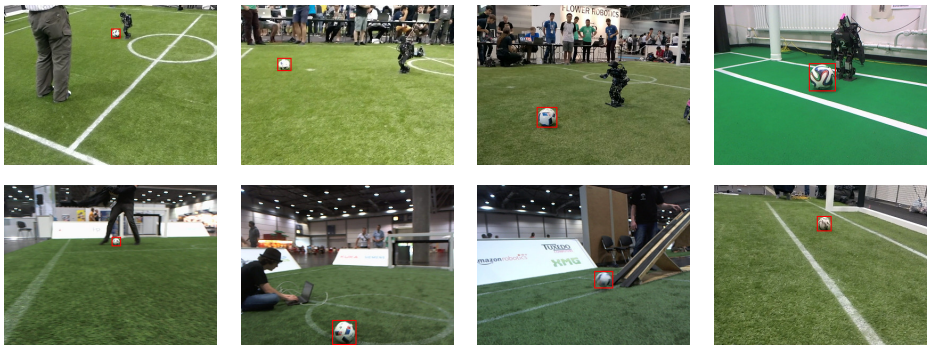

Fig. 1. Images taken from **training dataset** including their bounding boxes (red rectangles). There are 4 different types of balls in total. Most commonly recorded ball type is the Euro 2016 ball, which was the official one in Humanoid Kid-Size League at RoboCup 2017, Nagoya, Japan.

Fig. 2. Images taken from **test dataset**. This dataset mostly covers the Euro 2016 ball and footage recorded from an actual game of the competition in RoboCup Humanoid Kid-Size League at RoboCup 2017, Nagoya, Japan. The whole encounter's footage is just included in the test dataset. The training dataset does not include any of the images of this game. Besides, the test dataset includes another 351 images recorded by the WF Wolves team from a location that is not covered at all in the training dataset.

2.2 Metrics

There are several approaches to evaluate object detection frameworks. We supply four different metrics: *Intersection over Union* (IoU; also called *Jaccard index*[5]), *precision*, *recall*, and *radius accuracy*.
For IoU we give the average over the whole test dataset and also the 90th and 99th percentile since the intersection for false positives or false negatives is an empty set, thus heavily affecting the total IoU over the whole dataset. Providing the 90th and 99th percentile is a better measure for the accuracy of pixel-level detection for true positives.
For precision and recall we measure true positives (TP), false positives (FP), and false negatives (FN) with strong restrictions: if the models output contains multiple balls, we only extract the prediction with the highest activation. The center of this predicted ball cluster has to be within the ground truth, i.e. within the original label (ball pixels), to be counted as a TP. Effectively this means that at least 50% of such a predicted ball's pixels have to intersect with the ground truth ball label, otherwise it is counted as FP. If no significant cluster can be found in the model's output, then it is counted as a FN.
The fourth metric we use is radius accuracy. We propose this metric to allow to compare other models to ours that, for example, work on absolute coordinates and cannot produce pixel-level predictions to allow for IoU or other metrics. We hope this allows for comparability with as many models as possible. The radius accuracy is a radial error function. We compute the ball's predicted center and measure whether this point lies within a certain radius r around the ground truth (label). Formally, the accuracy with respect to a certain radius r is the sum (see Equation 1) over a scoring function f_r (see Equation 2) that measures if the squared difference between a prediction p and a label (ground truth) l is lower than the square of the radius for every image.

$$accuracy_r = \frac{1}{n} * \sum_{i \in I} f_r\Big((p_i^x - l_i^x)^2 + (p_i^y - l_i^y)^2\Big) \,. \tag{1}$$

$$f_r(x) = \begin{cases} 1 & x < r^2 \\ 0 & x >= r^2 \end{cases} \tag{2}$$

3 Proposed Architecture

Two architectures for neural networks and their implementation in Tensorflow are provided and evaluated against our dataset. Other teams are welcome to use these to compare their own results or improve our proposed architectures.

3.1 Model 1 (CNN)

This architecture is an updated version (see Figure 3) of the ball detection CNN model proposed by us at the 20th RoboCup International Symposium, 2016 in

[5] `https://en.wikipedia.org/wiki/Jaccard_index`

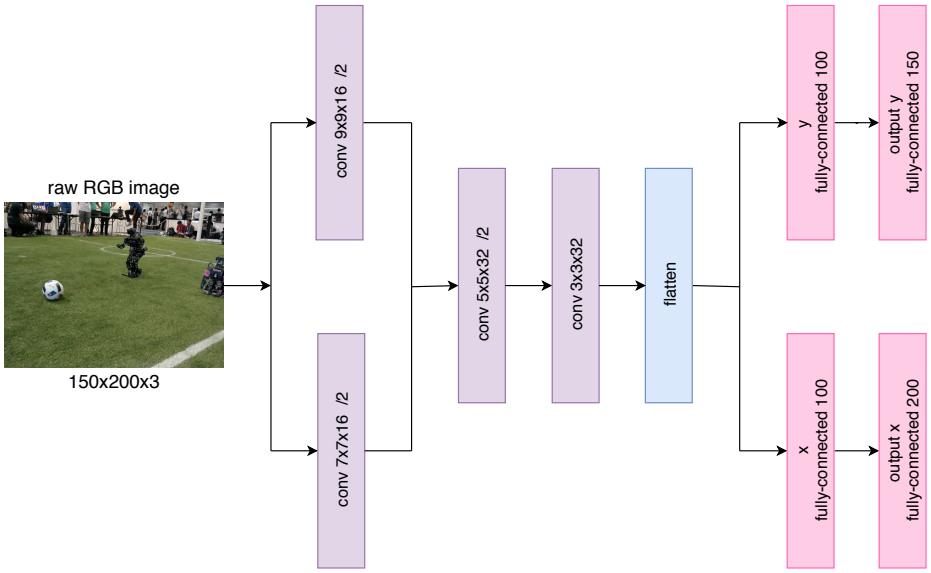

Fig. 3. Illustration of our proposed Model 1 (CNN). 7×7 and 9×9 convolutions are applied in parallel to the raw RGB input image. The second and third layer use 5×5 and 3×3 convolutional kernels respectively, before the information gets flattened and propagated to the fully-connected output channels to build probability distributions for x- and y-dimension. Strides of 2 are applied in the first two layers to reduce dimensionality.

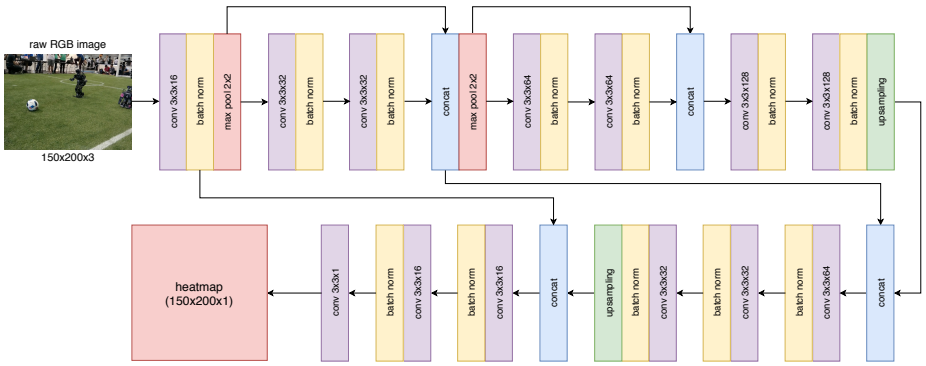

Fig. 4. Illustration of our proposed Model 2 (FCNN). Convolutional layers (purple), batch normalization (yellow), concatenation (so-called "skip connections"; blue), 2D upsampling (green), 2D max-pooling (red).

Leipzig [13]. Instead of soft-sign activation, we used leaky ReLU (rectified-linear units) activation, which showed reasonable results for our architecture [9]. The training procedure (teaching signal) stays the same as in the original paper.

Evaluation This architecture outputs two different probability distributions that should model a normal distribution where mean μ is expected to be at the ball's center in x- (first distribution) respectively y-dimension (second distribution). We did not fully utilize the power of the probability distribution (expensive post-processing) in this paper, because this model is already similar when it comes to computational complexity compared to Model 2 (FCNN). We simply used an argmax on the output to find the single neuron with the highest activation and take this as a prediction for the ball's center in x- & y-dimension. Further post-processing of the model's output, i.e. analyzing the probability distributions, would cover better results, but also would be done mostly on CPU and use up more computation time. We tried to streamline the model to also run near real-time on a NVIDIA Jetson TX2.

3.2 Model 2 (FCNN)

We developed a fully-convolutional neural network (FCNN) using TensorFlow[6] inspired by the model Schnekenburger et. al. proposed in their paper on object detection with the Sweaty robot [12]. Due to the limited computational power of RoboCup Humanoid Kid-Size robots, we propose a model with smaller input (150×200 for height and width; original paper uses 512×640) to allow near real-time execution on our NVIDIA Jetson TX2 hardware. We also feed raw camera input instead of normalized images. An illustration of our model can be seen in Figure 4. We use 2D max-pooling for dimensionality reduction and 2D bilinear up-sampling in our architecture to get a smoother heatmap as output because strided transposed convolutional layers for up-sampling led to "checkerboard artifacts" in our heatmap for some input images [2]. Xu et. al. showed a thorough evaluation of activation functions, which we used as a basis and found Leaky ReLU (rectified linear units) to cover the best results for us [14]. This kind of activation function was proposed by Maas et. al. [7].
We experimented with different initialization techniques for the model. The most stable results (test accuracies after finished training varied by only 0.1%) were achieved with Glorot random normal initialization for the convolutional weights and an all-zero initialization for the biases [4]. A dropout rate of 0.5 is applied to all layers but the first and last layer [5]. Padding is always set to "same", i.e. one of the padding options in TensorFlow, in order to keep dimensionality between convolutional layers.
For training the network we compute ellipses out of our bounding box labels in order to get near pixel-precise labels as training feedback for Model 2.

[6] `https://www.tensorflow.org/`

Evaluation To extract the ball's center we apply several steps onto the heatmap output of Model 2. At first, we apply Otsu's method to binarize and threshold the image [11]. Afterward, OpenCV's contour-finding algorithm is applied to the binary image, which will return clusters for each "hotspot" in the original heatmap. To extrapolate the most significant cluster, i.e. highest activation of the network in the heatmap, we sum up the network's original output over the indices of each cluster. This procedure extracts the "strongest" activation in the heatmap, returning the most significant cluster of "ball pixels". Afterward, we extrapolate the center point for a cluster using OpenCV's moments function.

4 Experimental Results

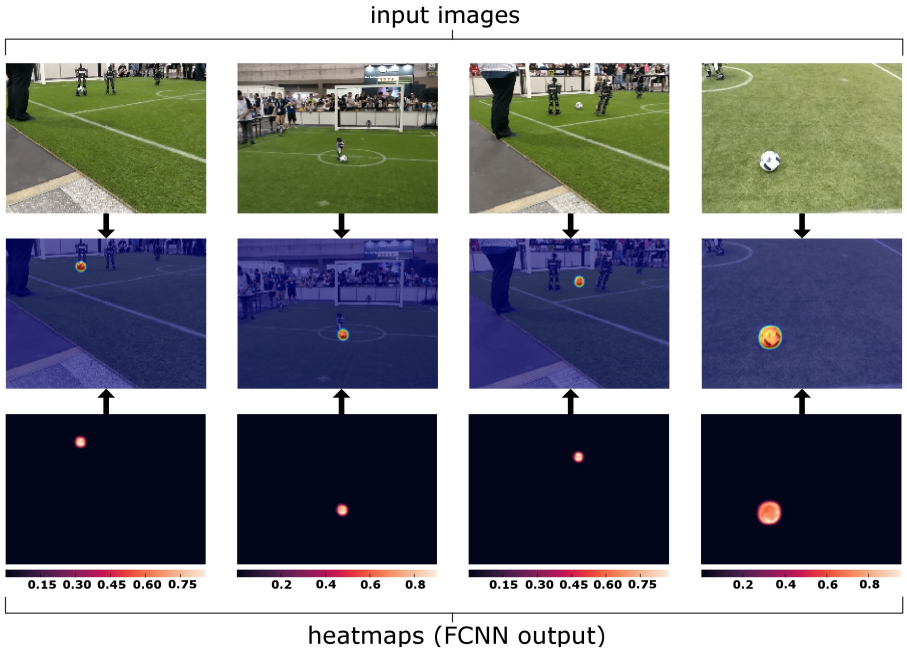

Fig. 5. Input images (test set images; top), **FCNN output** (heatmaps; bottom) and **combined plot** (center) for Model 2. The interval $[0.0, 1.0]$ is the possible range for the network's activation, hence this Figure shows that the network's neurons fire strongly for ball pixels and nearly homogeneously flat out to 0.0 otherwise.

4.1 Ball Localization

Table 1 shows that the fully-convolutional neural network (Model 2) has a very good ball localization quality throughout the test set, while the older model falls

Table 1. Results for full training (20 epochs).

	test dataset		robot test data
metric type	Model 1	Model 2	Model 2
radius 3 accuracy	30%	**93.3**%	39.9%
radius 5 accuracy	47%	**95.1**%	46.6%
radius 10 accuracy	55%	**96.3**%	63.9%
IoU (Jaccard index)	-	**74.3**%	43.9%
IoU 90th percentile	-	**90.6**%	88.1%
IoU 99th percentile	-	**95.7**%	93.7%
precision	-	**97.9**%	90.4%
recall	-	**98.3**%	86.6%

behind. Model 2 also delivers a reasonable performance on the robot test data set, which consists of 764 images recorded only on our robot at IRAN Open 2018. Our robot also walks and moves its camera leading to motion blur in the images.

4.2 False Positives

Additionally, we tested our models on a *negative* dataset from Leipzig, i.e. a dataset covering no ball at all. The dataset has 14,886 images covering different playfields from RoboCup 2016, Leipzig, recorded at different angles, heights, orientations, and so forth. Considering the size and complexity of the dataset it is a significant challenge to prove a model's robustness against false positives since any detection on this dataset can be considered a false positive. For Model 1 the 99th percentile of output activation showed values of 0.6 with a standard deviation of 0.2, leading to some false positives, even with post-processing applied to the output. Model 2's 99th percentile activation was at 0.003 with a standard deviation of 0.03. Actual balls in an image produce a mean activation of 0.7 (rounded) for Model 2, hence we apply a threshold at 0.5. In our negative dataset, only 1.04% of all images produce an output > 0.5. This results in 155 of 14,886 to falsely produce a positive output.

4.3 Hardware Benchmarks

The results for inference timings for a CPU and three different GPU types can be seen in Table 2. The NVIDIA Jetson TX2 was chosen since it is used by 7 teams [1] in HL and therefore the most used dedicated GPU. The Intel CPU is comparable to the performance of an Intel NUC which is used by 18 teams in the HL [1]. It shows that CPU inference timings per batch increase somewhat linear with batch size. Additionally, the CPU is considerably slow for many convolutional layers (which is expected, since a CPU has no specific hardware capability of speeding up these computations), rendering Model 2 more computationally

Table 2. Inference timings (mean values per batch through 1,000 runs)

Models	Hardware	GPU	Batch=1	Batch=4		Batch=8	
			total	per image	total	per image	total
Model 1	NVIDIA Jetson TX2	yes	0.041s	0.014s	0.057s	0.011s	0.089s
Model 2	NVIDIA Jetson TX2	yes	0.049s	0.028s	0.112s	0.023s	0.181s
Model 1	NVIDIA Titan X	yes	0.014s	0.004s	0.016s	0.002s	0.016s
Model 2	NVIDIA Titan X	yes	0.015s	0.005s	0.021s	0.004s	0.029s
Model 1	NVIDIA GTX 1080	yes	0.010s	0.003s	0.011s	0.002s	0.012s
Model 2	NVIDIA GTX 1080	yes	0.012s	0.005s	0.019s	0.003s	0.026s
Model 1	Intel Core i5 2500K	no	0.049s	0.025s	0.098s	0.026s	0.204s
Model 2	Intel Core i5 2500K	no	0.124s	0.130s	0.518s	0.136s	1.085s

expansive than Model 1 without a GPU. The GPU inference timings reveal that for larger GPUs (GTX 1080, Titan X) a mini-batch size of 1, 4 or 8 samples at once is too small to fully utilize the whole GPU. Thus, mean batch time does not increase linearly for these GPU models. Additionally, the Titan X is slightly slower than the GTX 1080 due to the 1080's higher GPU clock speeds. At training time with larger mini-batch sizes the Titan X is of course faster. The Jetson TX2 also performs noticeably well for both models. Model 1 has fewer layers (especially convolutional layers), hence it's computational complexity comes mainly through the fully-connected layers, which can be parallelized on the GPU for small batch sizes. This way the mean mini-batch timings for Model 1 do not increase heavily before mini-batch sizes of 8.

5 Discussion

Model 1 (CNN) was mainly selected to present a comparison to recent (2016) state-of-the-art models [13] challenged with new, more complex datasets. In comparison to most publications we mainly use footage from actual competitions and not just lab environments and in contrast to Model 1's original test data we now wanted to keep in mind the pending change of the rules that will double playfield sizes. To the best of our knowledge, our training and test dataset is the largest one publicly available for RoboCup. Due to the fact that we did not randomly split train/test data, but hand-picked playfields & games, or even locations in case of German Open footage, the test data is completely novel to the network. This, including our robot test dataset from Iran Open featuring only footage recorded *on* the robot, is more challenging than other test datasets, like the one used in [13]. Hence, it was expected that Model 1 (CNN) shows a considerably lower accuracy. The model performed well on older datasets, but struggles at ball localization on full-size playfields at very high distances with completely new environments, audiences, and so forth. However, despite the increased complexity of test data, this is partly also explainable with us not utilizing the probabil-

ity distribution output for x- and y-dimension, but just taking the maximum value's index of each distribution respectively. Further post-processing would definitely help to enhance Model 1's accuracy, but also greatly increase runtime performance. On the other hand, Model 1 performs better on CPUs due to its considerably lower amount of convolutional layers, which are only cheap to compute on GPUs. This emphasizes the usage of Model 1 or similar architectures for CPU-based robots, especially since post-processing of mentioned probability distributions is only greatly increasing runtime performance on comparably slow ARM-CPUs like the NVIDIA Jetson TX2. Intel NUC based robots, which are the standard for CPU-based robots, are much faster for the post-processing. Moreover, the lowest detection rates occur for distant balls (more than approx. 5m). Hence, at least for a goalkeeper (who does not need information about balls on the enemy half of the playfield) Model 1 might be a reasonable choice for fast detection of approaching balls on CPU-based architectures.

Model 2 (FCNN) covers a very high quality in ball localization overall. We wanted to push pixel-level accuracy with this model and supplied reasonable results considering the IoU results presented in Table 1. Model 2 scored 74.3% IoU on the whole test dataset. If we factor out some false positives for which the intersection is zero, hence greatly decreasing the total IoU, we get very high values of 90.6% IoU for the 90th percentile or 95.7% for the 99th percentile. Figure 5 illustrates this pixel-level ball localization. With a precision and recall of 97.9% and 98.3% respectively the results also show that Model 2 has a very low rate of false positives and false negatives. To further present the robustness of Model 2 we additionally benchmarked the model on another 14,886 images from RoboCup 2016, Leipzig, which do not contain any ball, thus any ball detection can be considered a false positive on this dataset. Even on this dataset, the false positive rate is just above 1% in total.

However, one might argue that the test dataset is not complex enough to challenge Model 2 since it does only over three different balls and does not include many very dark images. We included a robot test dataset recorded only on a robot walking over the playfield at Iran Open 2018 (see Table 1). Although the localization quality drops Model 2 delivers a reasonable performance, sufficient for ball localization with a precision of 90.4% and a recall of 86.6%. The drop mainly occurs due to the robot's walking: this introduces motion blur that heavily affects input image quality.

Considering runtime performance on GPUs it is also worth to think about batch processing since it is easier to parallelize batches on a GPU and thus achieve a higher overall utilization of the GPU's computing power. As shown in Table 2 the efficiency per image regarding runtime performance increases with higher batch numbers. Of course, we can not infinitely increase the batch size, because of (1) VRAM limitations, (2) reaching the maximum parallelization performance of the GPU, and (3) latency. On the contrary, it has to be considered that for distant balls at least latencies do not matter, but accuracy does. If the robot always processes batches of 8 images then even if the robot is walking the ball information will be more stable due to the post-processing of the batch. The

probability of 8 images being motion-blurred or covering other problems is lower than for single images.

6 Conclusion

We proposed a state-of-the-art deep learning architecture to detect & localize balls in complex RoboCup Humanoid Soccer scenes that is able to run in near real-time on NUC- or GPU-based robots, such as the NVIDIA Jetson TX2 based robots we use. We also offer the full training and test dataset, the robot test dataset and the additional negative dataset to test against false positives since we want to emphasize the development and comparison of deep learning architectures in RoboCup. To the best of our knowledge, our datasets are by far the largest and most complex ones publicly available in RoboCup. The results were achieved keeping the limited computational resources of robots of the Humanoid Kid-Size League in mind, thus more complex architectures might very well score even better results. However, we also wanted to push ball detection to a pixel-level localization with Model 2, which comes at the cost of needing a GPU or fast CPU. The models can train 20 full epochs ($20 \times 35,327$ images) in less than 4 hours on an NVIDIA TITAN X GPU when combined with a parallel data pre-loading algorithm, while also evaluating the test dataset after each epoch. We use a PCI-E SSD card for storing the images to ensure fast loading speeds, but we will also supply HDF5 files that speed up loading from hard disks with the only disadvantage of a larger total file size.

7 Future Work

We will release new versions of the dataset each year, to include new environments from each years RoboCup competitions and to increase the difficulty of the test dataset. A large dataset with complex test data eases the transition and prevents a drop of game quality similar to the year when the multi-colored ball was introduced. The datasets also include some, although not many, blurry images. Since a game of RoboCup soccer has to be dynamic recorded images often involve motion blur from the camera itself or simply due to the robot currently walking. Another problem is that different backgrounds, especially light sources, lead to vastly different color spaces. These distortions can heavily affect detection rates of deep learning architectures because the system will focus on learning significant features for ball detection, not de-blurring kernels. We are currently working on neural network based de-noising frameworks to reduce this kind of problems.
For fast moving balls, which will become more common in the future, we will also try to combine neural architectures like the FCNN for detection in combination with fast object tracking architectures. A correlation tracker, for example, is computationally cheap and might produce reasonable results if supplied with accurate regions of interest from an FCNN.

Acknowledgement

We would like to thank everyone from our local RoboCup Team, the Hamburg Bit-Bots, who helped with tagging the images. We are grateful to the NVIDIA corporation for supporting our research[7]. We used the donated NVIDIA Titan X (Pascal) to train our models. This research was funded by the German Research Foundation (DFG) and the National Science Foundation of China (NSFC) in project Crossmodal Learning, TRR-169.

References

1. Humanoid league team description papers (2018), https://www.robocuphumanoid.org/hl-2018/teams/, acessed: 09.03.2018
2. Aitken, A., Ledig, C., Theis, L., Caballero, J., Wang, Z., Shi, W.: Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize (jul 2017), http://arxiv.org/abs/1707.02937
3. Fiedler, N., Bestmann, M., Hendrich, N.: Imagetagger: An open source online platform for collaborative image labeling, private communication, submitted
4. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. Aistats **9**, 249–256 (2010). https://doi.org/10.1.1.207.2059
5. Hinton, G.: Dropout : A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research (JMLR) **15**, 1929–1958 (2014)
6. Humanoid Leauge Technical Committe: Humanoid league proposed roadmap (2014), https://www.robocuphumanoid.org/wp-content/uploads/HumanoidLeagueProposedRoadmap.pdf, acessed: 10.04.2018
7. Maas, A., Hannun, A., Icml, A.N.P., 2013, U.: Rectifier nonlinearities improve neural network acoustic models. pdfs.semanticscholar.org https://pdfs.semanticscholar.org/367f/2c63a6f6a10b3b64b8729d601e69337ee3cc.pdf
8. Menashe, J., Kelle, J., Genter, K., Hanna, J., Liebman, E., Narvekar, S., Zhang, R., Stone, P.: Fast and precise black and white ball detection for robocup soccer
9. Mishkin, D., Sergievskiy, N., Matas, J.: Systematic evaluation of CNN advances on the ImageNet (jun 2016), http://arxiv.org/abs/1606.02228
10. O'Keeffe, S., Villing, R.: A benchmark data set and evaluation of deep learning architectures for ball detection in the robocup spl (2017)
11. Otsu, N.: A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics **9**(1), 62–66 (jan 1979). https://doi.org/10.1109/TSMC.1979.4310076, http://ieeexplore.ieee.org/document/4310076/
12. Schnekenburger, F., Scharffenberg, M., Wülker, M., Hochberg, U., Dorer, K.: Detection and Localization of Features on a Soccer Field with Feedforward Fully Convolutional Neural Networks (FCNN) for the Adult-Size Humanoid Robot Sweaty. Proceedings of the 12th Workshop on Humanoid Soccer Robots, 17th IEEE-RAS International Conference on Humanoid Robots pp. 1–6 (2017)
13. Speck, D., Barros, P., Weber, C., Wermter, S.: Ball Localization for Robocup Soccer using Convolutional Neural Networks. In: RoboCup 2016: Robot World Cup XX, pp. 19—-30 (2017)
14. Xu, B., Wang, N., Chen, T., Li, M.: Empirical Evaluation of Rectified Activations in Convolutional Network (may 2015), http://arxiv.org/abs/1505.00853

[7] https://developer.nvidia.com/academic_gpu_seeding