JET-Net: Real-Time Object Detection for Mobile Robots

Bernd Poppinga[1,2] and Tim Laue[1]

[1]Universität Bremen, [2]JUST ADD AI GmbH
{poppinga,tlaue}@uni-bremen.de

Abstract. In most applications for autonomous robots, the detection of objects in their environment is of significant importance. As many robots are equipped with cameras, this task is often solved by image processing techniques. However, due to limited computational resources on mobile systems, it is common to use specialized algorithms that are highly adapted to the respective scenario. Sophisticated approaches such as Deep Neural Networks, which recently demonstrated a high performance in many object detection tasks, are often difficult to apply. In this paper, we present JET-Net (**J**ust **E**nough **T**ime), a model frame for efficient object detection based on Convolutional Neural Networks. JET-Net is able to perform real-time robot detection on a NAO V5 robot in a robot football environment. Experiments show that this system is able to reliably detect other robots in various situations. Moreover, we present a technique that reuses the learned features to obtain more information about the detected objects. Since the additional information can entirely be learned from simulation data, it is called *Simulation Transfer Learning*.

1 Introduction

Depending on the application domain, a mobile robot needs to detect different objects with a certain reliability and precision. In the RoboCup Standard Platform League, to which our work has been applied, two important categories of moving objects are the ball and the other robots. For the ball, multiple sophisticated detection techniques, mostly based on machine learning, already exist [24,18,14]. However, the reliable detection of other robots is a much more complex problem fur multiple reasons: a robot is not fully symmetric and thus looks different depending on the angle of view, it can have different postures (e.g. standing, lying, or getting up), and, especially when it is close, often only parts of a robot are in the current field of view. Futhermore, changing lighting conditions might make it difficult to use the robot's colored jersey as a reliable cue. Nevertheless, some highly spezialized robot detection approaches have already been implemented and are currently in use, as described in Sect. 2.2.

In recent years, detecting objects by using Deep Neural Networks has been very successful in many domains, some examples are given in Sect. 2.1. These approaches are able to robustly generalize over deviations in perspective, shape, and lighting conditions. However, to achieve a certain level of robustness and precision, many network layers are required, resulting in a high demand of computing power, which is not available on many mobile platforms such as the NAO V5, which was the target platform for our development and which also needs to carry out many other tasks that require computing time.

The main contribution of this paper is an adaptable network architecture that is able to perform object detection on computationally limited robot platforms. Although the architecture itself is kept general, its performance is demonstrated for the task of robot detection, which it is able to carry out with a precision and robustness that is suitable for robot football. In comparison to most related works, the network carries out the full detection process and does not only classify preselected candidate regions. An additional contribution is an extension to the main network that makes it possible to not only perform object detection but also to learn additional properties of the objects entirely from the simulation. This is demonstrated by learning the distances to detected robots.

The remainder of this paper is organized as follows: First, Section 2 discusses related approaches for object detection by Deep Neural Networks in general as well as for object detection in the RoboCup Standard Platform League domain in particular. Afterwards, a description of our approach is given in Section 3. An evaluation of the accuracy and the performance of our approach is presented in Section 4. Finally, the paper concludes in Section 5.

2 Related Work

2.1 Object Detection by Deep Neural Networks

There are many recently published works regarding object detection, most of them aim for two major objectives: a high mean average precision (MAP) and a low inference time. Published in 2015, Faster R-CNN [23] was the state of the art in terms of MAP for a long time. This is due to its two-staged architecture, which comes at the costs of increased runtime. Furthermore, Faster R-CNN has introduced anchor boxes, which are used until today. The Single Shot MultiBox Detector (SSD) [13] approach takes many aspects of Faster R-CNN and puts them into a one-staged detector by removing the preselection step. This means that the classifier has eventually to deal with many simple negatives that take away the focus form the hard examples. To compensate these disadvantages, the hard negative mining was presented, which allows the detector to concentrate on the difficult examples. In contrast to SSD, the YOLO approach [20,21] is optimized to reach extraordinary inference time. This approach has been iteratively improved so that the latest version YOLOv3 [22] offers a good trade-off between MAP and inference time. In 2017, the Retina Net [12] one-staged detector made it finally possible to outperform two-staged detectors by applying a technique called Focal Loss. Additionally, it uses a feature pyramid architecture which presents the current state of the art [11].

2.2 Object Detection in RoboCup Soccer

In recent years, Deep Learning has already been successfully used for the detection of the Standard Platform League's new ball [24,18,14]. A similar approach is also used by [1] for detecting robots. However, all these approaches have in common that they use neural networks for classifying previously computed candidate regions and thus depend on other software components. In the RoboCup Humanoid League, a full detection approach, which would not have real-time inference on a NAO robot, has been presented by [25].

In addition to these machine learning approaches, many other solutions exist that heavily rely on the particular design of the robot football environment and a robust color classification. By knowing that everything takes place on an even green floor, objects can be detected by just finding gaps in green [5,10]. In the SPL, the colors of the robots as well as their jersey colors are known, this makes a direct detection based on a previously conducted color classification possible, as described, for instance, by [15] and [2]. This approach is also applied in some other leagues, for instance in the Small-Size Robot League's SSL-Vision [27].

However, due to changes in the RoboCup rules that require more natural lighting conditions, these approaches become less maintainable und applicable by the time. One recently published approach by [9] therefore combines basic robot vision techniques on grayscale images with candidate classification through a Convolutional Neural Network to detect all objects on an SPL field.

3 JET-Net: The Just Enough Time Approach

In this section, we explain JET-Net's model design, followed by the applied training. Then we show how we use JET-Net for robot detection in the SPL. In the end, we explain how to use *Simulation Transfer Learning* to predict more characteristics while only using data from simulation.

3.1 General Model Design

Even though we want to apply the object detection to the specific task of robot detection in the SPL, JET-Net provides a general framework that can be adapted to different tasks. Therefore, we do not use any special features of the NAO as in [17] and restrict ourself to the size and ratio of the object. However, within these restrictions, we want to make everything as task-specific as possible. That means that we adapt our resolution to the camera and computational power of the robotic system. Furthermore, we do not predict thousands of classes but one or two. Whenever there is a design question, we always take just enough to fulfill the task. This results in an efficient architecture that needs just enough time to do its task, without any significant overhead. Our model design frame is used to define a basic architecture, which is optimized afterwards.

Model Frame For the model design, we define a frame that leads the design process. This frame consists of three units. We start by using alternating *Feature Modules* and *Scale Modules* to get features of different hierarchy levels. In the end, we add a layer for the bounding box prediction, which uses anchor boxes as introduced in [23].

- **Feature Module:** This module uses stacked 3x3 convolutions to extract the important features. The number of layers per module and the number of filters are hyperparameters. Each *Feature Module* starts with a Batch Normalization.
- **Scale Module:** This module follows directly after every *Feature Module*. It reduces the image size by a factor of two. We tried different approaches for the scaling. As we have a natural lack of parameters in small neural networks, we choose 3x3 convolutions with a stride of 2 over 2x2 convolutions and Max Pooling. However, the number of filters will remain a hyperparameter.
- **Box Prediction:** This very last network layer predicts the bounding boxes. Here, 1x1 convolutions are used to determine the five box parameters for every cell in the remaining image. The number of anchor boxes per cell is a hyperparameter here.

Speed Up Even though one should choose a minimal input size and a lightweight model design to achieve a fast inference, we are not satisfied by the efficiency of normal convolutions. For this purpose, we chose the MobileNet [6] approach over other techniques such as SqueezeNet [7]. Admittedly, SqueezeNet reduces the number of parameters, but there is hardly a faster inference. Therefore, its usage would be rather counterproductive, as our small network has a natural lack of parameters. This is why we go with the MobileNet approach and replace some convolutions with *Separable Convolutions*.

For 3x3 convolutions, one can save for example 15.3% of the calculations when using 24 filters like we did for our robot detector. However, it is not always advisable to use Separable Convolution. When working with grayscale images, we only have one channel at the input layer and the use of Separable Convolution would lead to an increase in calculations. The same goes for the box prediction layer, where 1x1 convolutions are applied. Furthermore, since layers deeper in the network operate on smaller images, the acceleration has not such a big influence. Therefore, it is sometimes worth to choose normal convolution in those layers to get more parameters while retaining a good inference time.

3.2 Training

As usual for this kind of object detector, we used two different loss functions for training. Additionally, we present our augmentation steps and show, how we can achieve even more speed up the inference time by applying pruning.

Loss Functions The task of finding bounding boxes for objects in the image consists of two subtasks: Determining if a certain area shows a wanted object and finding the exact box. For the corresponding training, we propose the following loss functions:

MACE Loss Each anchor box needs to be classified whether it contains an object whose bounding box needs to be found or not. This is not a trivial task. As we have so many potential anchor boxes, many of them are quite easy to classify as negatives. This causes the problem that the optimization does not concentrate on the hard examples. But we actually care more about those that are hard to classify. This is also the big advantage of two-staged detectors, as they filter the easy examples in the first step, so that they only need to classify the hard examples in the second stage. Other one-staged detectors already tried to deal with this problem: SSD proposed hard negative mining, where only a fixed number of the hardest examples are used to calculate the loss. The *Focal Loss* approach introduces a weighting which weights bigger errors higher. For a vector of errors between label and prediction y_{err}, (Equation 1) shows how the Focal Loss is calculated. We took that approach but replaced the negative logarithmic error by the mean squared error. This allows to sum up the equation into $i^{2+\gamma}$. Furthermore, we discovered that $\gamma = 1$ works quite well for our tasks. Therefore, we will refer to the used loss function (Equation 2) as mean absolute cubed error (MACE) .

$$\text{FocalLoss}(y_{err}) = \frac{1}{|y_{err}|} \cdot \sum_{i \in y_{err}} -log(1-i) \cdot i^{\gamma} \tag{1}$$

$$\text{MACE}(y_{err}) = \frac{1}{|y_{err}|} \cdot \sum_{i \in y_{err}} i^2 \cdot i^1 = \frac{1}{|y_{err}|} \cdot \sum_{i \in y_{err}} i^3 \tag{2}$$

IoU Loss The loss, which rates the position and size of the bounding box, is responsible for the other four parameters of the bounding box: x-position, y-position, width, and height. Other approaches use a variant of the absolute distance between the wanted values and the predictions (Faster R-CNN) or the relative distance (SSD). We propose a more abstract measurement of the loss by using the mean IoU (Intersection over Union) value as the loss function. We argue that those four values do not contribute equally to a good matching bounding box. However, the IoU value does take all aspects into account and returns one value.

$$\text{IoU-Loss}(true, pred) = 1 - \text{MEAN}(\text{IoU}(true_{box}, pred_{box})) \tag{3}$$

Augmentation To achieve a better generalization, we use data augmentation. While some publications propose an exhausting approach for augmentation, we concentrate on creating only images that could actually appear in reality. This ensures that we don't use capacities for images that will never appear. This leaves us with the following procedures:

Layertype	Filter b.p.	Filter a.p.	Filter Size	Strides	Padding	Output
BNorm	-	-	-	-	-	80x60
Conv	24	16	3x3	1	Same	80x60
SConv	24	24	3x3	2	Same	40x30
BNorm	-	-	-	-	-	40x30
SConv	24	16	3x3	1	Same	40x30
SConv	24	20	3x3	1	Same	40x30
SConv	24	20	3x3	2	Same	20x15
BNorm	-	-	-	-	-	20x15
SConv	24	20	3x3	1	Same	20x15
SConv	24	20	3x3	1	Same	20x15
SConv	24	24	3x3	1	Same	20x15
SConv	24	24	3x3	2	Same	10x8
BNorm	-	-	-	-	-	10x8
Conv	24	24	3x3	1	Same	10x8
Conv	24	24	3x3	1	Same	10x8
Conv	24	24	3x3	1	Same	10x8
Conv	24	24	3x3	1	Same	10x8
Conv	20	20	3x3	1	Same	10x8
Inference time	*12.0 ms*	*9.0 ms*				

Table 1. Final model design for robot detection on a NAO V5. The table shows our JET-Net instantiation for robot detection. Filter b.p. means before pruning and a.p. after pruning, respectively. The inference time has been measured on a NAO V5 robot.

- **Zoom:** For the zoom operation, the image size is increased by a factor between 1.0 and 1.5. Since the input size remains untouched, we can now pick a cutout from the zoomed image.
- **Flip:** Since the playing field and the robots are symmetrical, we can flip the image at the vertical axis. A flip at the horizontal axis is not performed, as it would produce impossible images.
- **Ground Truth Box Noise:** To apply more noise, the borders of the ground truth boxes are varied. This makes it way harder to memorize certain boxes. To vary the boxes, each border line is moved in a certain area ($\pm 5\%$).

Pruning Another way of speeding up calculations is to not execute them in the first way. For that reason, we use pruning to remove those filters that contribute the least. In [16], many pruning methods are compared. Although the normed activation was not the best pruning method, it was still a very good one and is comparatively easy to implement and calculate. A crucial part of the pruning is the fine tuning between the different pruning runs to compensate the damage. This leaves us with an iterative algorithm that looks for the filter with the least activation, removes it, and treats the dealt damage with fine tuning. This

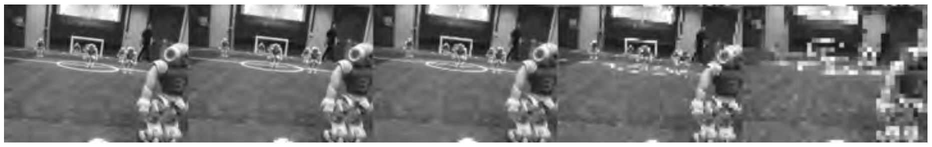


Fig. 1. Sampling Levels. For the robot detection application, we tried different sampling levels from 640x480 (outer left) to 40x30 (outer right). Each images shows a quarter of the pixels used in the image before. We chose 80x60 (second from right) for input, as it is the lowest resolution that still shows every robot.

can be repeated until a certain size is reached or the loss drops under a certain threshold. As the number of filters in the box prediction unit is fixed, we excluded this from the pruning process. The resulting filter counts can be seen in Table 1. The pruning allowed us to reduce the inference time from 12.0 ms to 9.0 ms.

3.3 Application: Just Enough Robot Detection

For the scenario of the SPL, we created an instantiation of our model frame. As we use the current B-Human software stack [24], the inference on the robot is performed by B-Human's JIT compiler [26], which is able to process four convolutional filters at a time and which can process up to 24 filters fast. Our proposed network takes this into account. Furthermore, we take advantage of the fact that convolutions are cheaper on smaller images, as they don't need to be applied so often. The final model is described in Table 1. For the training, we used images from both sources, reality as well as simulation. As the B-Human framework provides images with a resolution of 640x480 pixels in YUV422 color space, some preprocessing is needed before we can pass the data to the network:

- **Color space conversion:** As multiple ball detectors for this domain have already shown, the use of only one channel, the Y channel, is enough to achieve good results and contributes to more robustness.
- **Normalization:** Because mobile robots might deal with changing and challenging lighting conditions, a 2% min-max normalization is applied.
- **Subsampling:** Object classifiers such as the ball detectors often classify 32x32 pixels or less. However, in this scenario, we deal with the whole image instead of small patches. Therefore, we scale the image to an input size of 80x60 pixels (see Figure 1).

3.4 Simulation Transfer Learning

When we look from a more global point of view, a further possibility for optimization appears: Instead of just improving the inference time, we can reuse the features created for the detection task and use them to estimate further characteristics of the detected object, such as the distance. However, labeling those

GAN-Training	Discriminator Accuracy Both sources	Sim only
before	75,7%	58,4%
after	64,2%	58,2%

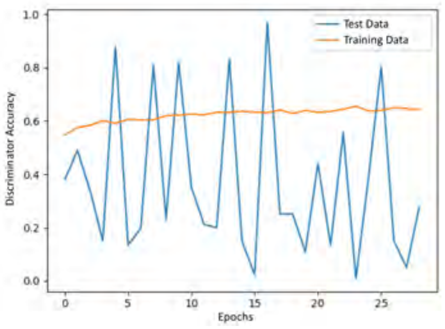


Fig. 2. GAN-Training. The table shows the training results before and after the GAN training. You can see that only the use of images from both sources provides us with us a high level of anonymization. While the discriminator is able to find some rule for the real images of the training data to tell them apart, it is more or less randomly guessing for the simulation data in the training data. And finally, the discriminator has no clue for the test data.

features is an annoying and exhausting task. Therefore, we want our network to use features for the robot detection that exist in the reality as well as in the simulation. To achieve this, we train our network not only with real images but also with images generated by a simulation. This forces the network to learn features for both sources. Those features could end up in two separated sets: one for real images and one for artificial images. However, as our network is so small, it already learns some features that appear in both sources. We finalize those features by applying a form of training that is similar to Generative Adversarial Network (GAN) training [4]. For this, we define the whole network, except for the box prediction layer, as the generator and create a discriminator which is almost as powerful as the box prediction layer (see Figure 3). Afterwards, we alternatingly train the discriminator and the box prediction layer with the generator. This results in a state, where the network cannot tell apart the simulated images and the real ones. When we reach this state, we can learn additional feature, like the distance, by using simulated images only. To do this, we freeze the generator and ignore the loss of the real images for the additional features, as we don't have valid labels for them. In this way, the real images are only used for the bounding boxes while we can obtain other characteristics from the simulation. In Figure 2, you can see that even before the GAN training the learned features deliver almost no information from which we can derive the source. Only through overfitting on the training data, the discriminator is somehow able to label some of the images right. These are promising results, which imply that it could be possible to use these features to learn more characteristics only from simulation images as almost every information about the image source is vanished. We refer to this technique as *Simulation Transfer Learning*.

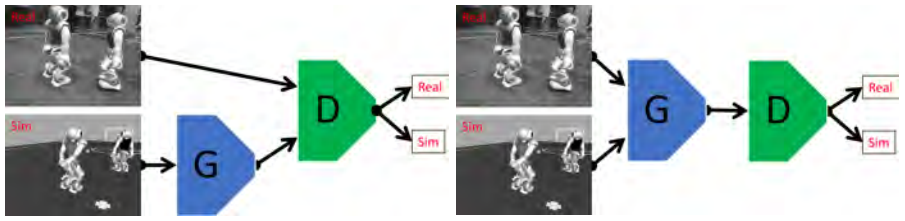


Fig. 3. GAN Architectures. The left image shows a typical GAN architecture. The image from the simulation is processed by the generator (G) to look like a real image. Afterwards, both images are passed to the discriminator (D) which tries to distinguish both real images and simulation images. On the right side, our adapted architecture is shown: We send images from both sources through the generator (G), as we just want them to look alike.

4 Evaluation

For the evaluation of our approach, we measure the performance of the robot detector. Therefore, we trained the proposed model (Table 1) with 27.000 publicly available images from ImageTagger [3] and 28.000 images generated in simulation by SimRobot [8][1]. Additionally, we used the simulated images to gather the distances of the robots, which were used for Simulation Transfer Learning. We do this by executing two experiments: a static setup and a dynamic one.

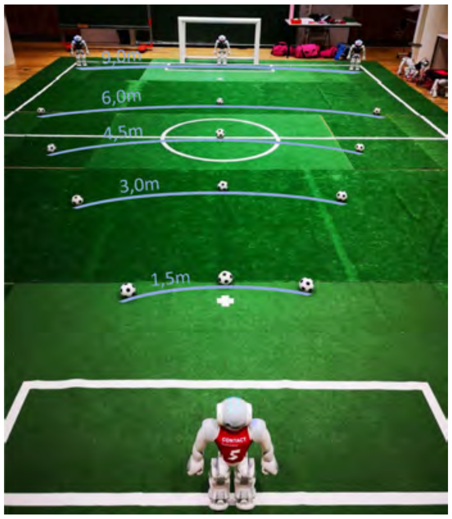
	Detected Robots		Distance Mean		Distance Std Deviation	
Distance	JET-Net	B-Human	JET-Net	B-Human	JET-Net	B-Human
0.3m	**2/2**	0/2	-	-	-	-
1.5m	**9/9**	8/9	**1.4** m	1.2 m	0.2 m	**0.1** m
3.0m	8/9	8/9	**3.0** m	2.7 m	0.5 m	0.5 m
4.5m	**7/9**	1/9	**5.1** m	3.7 m	0.6 m	0.6 m
6.0m	**1/9**	0/9	-	-	-	-
9.0m	0/9	0/9	-	-	-	-

Table 2. Results of the static evaluation. The column *Detected Robots* shows how many of the nine poses have been recognized. *Distance Mean* and *Standard Deviation* show the results of the distance prediction through all poses.

4.1 Different Distances and Poses

For evaluating the static setup, we placed one of our robots at the end of the field. Then we defined five distances (1.5 m, 3 m, 4.5 m, 6 m, 9 m) and three different

[1] The generated images as well as a script for downloading the datasets that we used from ImageTagger are available online at `https://sibylle.informatik.uni-bremen.de/public/JET-Net/`



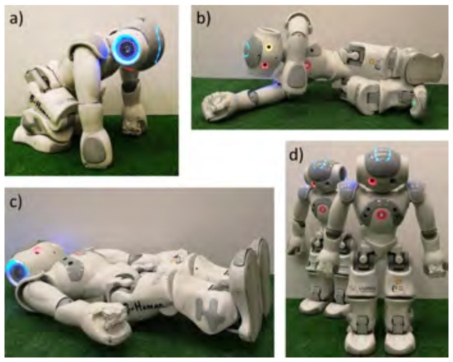


Fig. 4. Evaluation poses. Besides some standard poses like standing, we tried four special poses. a) gorilla pose b) keeper jump c) lying d) two robots.

Fig. 5. Overview of the evaluation setup. We placed the robot that has to be detected on one the positions marked by balls after the other. The robot in the red jersey has to detect it.

angles for each distance (left, center, right). For each distance, we tested nine different poses, most of them used for the center position: robot stands frontal (1), robots stands sidewards (2), like 1 but at left position (3), like 1 but at right position (4), gorilla position (5), robot lies on the floor (6), like 1 but with ball in front of the robot (7), two robots standing behind each other (8), after goalkeeper jump (9). Some poses are shown in Figure 4, Figure 5 provides an overview of the setup. Furthermore, we evaluate pose 1 and 2 also for 0.3 m to evaluate close-ups. For each distance and pose, we checked, if the robot is detected as well as the error of the estimated distance. For more meaningful results, we compare our robot detector to the B-Human's current implementation [24].

The results show that our proposed network detects the robot in every pose in the area until 4.5 m. Only the lying robot is problematic from 1.5 m on, which is probably because its representation in the box prediction layer is too small. Until the distance of 1.5 m it is also possible to tell the two robots from pose 8 apart. Starting from the distance of 6 m almost no robot is recognized. The B-Human detector in comparison is able to detect robots in a distances between 1.5 m and 3 m independently from the pose. However, lying robots are often detected as two robots. The B-Human detector calculates the distance to a robot by detecting a robot's feet and then incorporating the knowledge about its camera's perspective. Within an area of 3 m the mean of the estimated distance over all poses of our approach is more accurate than the B-Human solution. Which has, on the other hand, a smaller variance. An overview of the results is given in Table 2.

	Recall			Precision	IoU			MAP
approach	total	until 6m	until 3 m	total	total	until 6m	until 3 m	total
B-Human	0.204	0.285	0.583	**0.972**	0.657	0.657	0.662	-
JET-Net	**0.662**	**0.781**	**0.929**	0.856	**0.683**	**0.705**	**0.793**	0.591

Table 3. Results of the dynamic evaluation. Comparing JET-Net to B-Human. As B-Human does not provide probabilities, there is no MAP to be calculated.

4.2 Robot Detection During a Game

We also evaluated the performance for a test game of B-Human. During the game, we measured the precision, recall, IoU, and MAP for both approaches. As the B-Human approach does not rank the detections, there is no MAP value for this approach. While the precision of B-Human is still a bit better than our approach, we outperform the B-Human approach in every other category. Especially, we are able to achieve a much higher recall (see Table 3). The B-Human detector is tuned to strongly prefer precision over recall.

4.3 Performance

The software on the NAO robot alternatingly processes the image of the upper and the lower camera. As each camera runs at 30 fps, this results in an overall 60 fps. When processing the upper image, which we use in a higher resolution than the lower one, the B-Human detector only needs 0.5 ms per frame on average but relies on a previously executed color segmentation that takes more than 3 ms. While the main task of the upper camera is the raw robot detection, the lower camera is mainly used for calculating the robot distances. As shown before in Table 1, JET-Net takes 9.0 ms for the inference of one image on a NAO V5 robot. However, as our network is able to predict the distances only from one image, the lower camera is not needed anymore, which can be considered as a reduction of the average runtime to 4.5 ms. This makes it still possible to run each camera at 30 fps. Furthermore, recent measurements on a new NAO V6 robot revealed a runtime of only 2.5 ms.

5 Conclusion and Future Work

In this paper, we presented a framework for designing and training object detectors for mobile robots that are real-time capable, which is called JET-Net. By defining a coarse structure that only depends on few hyperparameters, we provide an easily adaptable network with advice for finding those few hyperparameters. We used that framework for instantiating a robot detector for the RoboCup SPL that runs on a NAO V5 in real-time. Furthermore, in comparison to the current robot detector of B-Human, we are able to recognize three times more objects by only losing a few percents in precision. As our system delivers probabilities, this ratio can be shifted to the one or the other direction. When comparing the maximum distance to which a robot can be detected, we are able to detect robots for further 1.5 m compared to B-Human's approach.

Additionally, we presented Simulation Transfer Learning, a technique that makes further use of the features that are already created for the robot detection. In doing so, it is able to receive more information about a detected object with almost no further calculations. The reuse of the already learned features makes it possible to learn these additional features from simulation only. This provides a simple method for crossing the simulation reality gap. In our experiments, we used this approach for estimating the distance to detected robots. The results show that in general, we can provide a better measure than current approaches that calculate the distance regarding to the current pose of the robot. However, this depends strongly on the pose of the detected robot. Thus, we have a bigger standard deviation than the geometric approach.

Further works can investigate the detection of other objects like a ball or goal posts in the SPL or entirely new scenarios on other robot platforms. As we did not use any special characteristics of the NAO V5, our approach should be easily transferable to these scenarios. Moreover, the computation of other characteristics than the distance, like the orientation, could be obtained from the simulation and applied to real world scenarios. Although the inference time of JET-Net is already satisfying, it could probably be improved by using approximations techniques like XNOR-Net [19].

Acknowledgements

We would like to thank the members of the team B-Human for providing the software framework for this work as well as everybody who contributed labeled data to the ImageTagger platform, especially the Nao Devils team.

References

1. Cruz, N., Lobos-Tsunekawa, K., Ruiz-del Solar, J.: Using convolutional neural networks in robots with limited computational resources : Detecting nao robots while playing soccer. In: RoboCup 2017: Robot World Cup XXI. Lecture Notes in Artificial Intelligence, vol. 11175, pp. 19 – 30. Springer (2018)
2. Fabisch, A., Laue, T., Röfer, T.: Robot recognition and modeling in the robocup standard platform league. In: Proceedings of the Fifth Workshop on Humanoid Soccer Robots in conjunction with the 2010 IEEE-RAS International Conference on Humanoid Robots. Nashville, TN, USA (2010)
3. Fiedler, N., Bestmann, M., Hendrich, N.: ImageTagger: An open source online platform for collaborative image labeling. In: RoboCup 2018: Robot World Cup XXII. Springer (to appear)
4. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Networks pp. 1–9 (jun 2014), `http://arxiv.org/abs/1406.2661`

5. Hoffmann, J., Jüngel, M., Lötzsch, M.: A vision based system for goal-directed obstacle avoidance. In: RoboCup 2004: Robot Soccer World Cup VIII. Lecture Notes in Artificial Intelligence, vol. 3276, pp. 418–425. Springer (2005)
6. Howard, A.G., Wang, W.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861 (2017)
7. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size pp. 1–5 (2016), http://arxiv.org/abs/1602.07360
8. Laue, T., Spiess, K., Röfer, T.: SimRobot - A General Physical Robot Simulator and Its Application in RoboCup. In: RoboCup 2005: Robot Soccer World Cup IX. Lecture Notes in Artificial Intelligence, vol. 4020, pp. 173–183. Springer (2006)
9. Leiva, F., Cruz, N., Bugueño, I., Ruiz-del Solar, J.: Playing soccer without colors in the SPL: A convolutional neural network approach. arXiv preprint arXiv:1811.12493 (2018)
10. Lenser, S., Veloso, M.: Visual sonar: Fast obstacle avoidance using monocular vision. In: Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003). vol. 1, pp. 886–891. Las Vegas, USA (2003)
11. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 2017-Janua, 936–944 (2017)
12. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal Loss for Dense Object Detection (aug 2017), http://arxiv.org/abs/1708.02002
13. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. Lecture Notes in Computer Science, vol. 9905, pp. 21–37 (2016)
14. 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. In: RoboCup 2017: Robot World Cup XXI. Lecture Notes in Artificial Intelligence, vol. 11175, pp. 45 – 58. Springer (2018)
15. Metzler, S., Nieuwenhuisen, M., Behnke, S.: Learning visual obstacle detection using color histogram features. In: RoboCup 2011: Robot Soccer World Cup XV. Lecture Notes in Computer Science, vol. 7416, pp. 149–161. Springer (2011)
16. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning Convolutional Neural Networks for Resource Efficient Inference (2015), 1–17 (2016), http://arxiv.org/abs/1611.06440
17. Mühlenbrock, A., Laue, T.: Vision-based orientation detection of humanoid soccer robots. In: RoboCup 2017: Robot World Cup XXI. Lecture Notes in Artificial Intelligence, vol. 11175, pp. 204 – 215. Springer (2018)
18. Nao-Team HTWK: Team research report 2018 (2019), https://htwk-robots.de/documents/TRR_2018.pdf
19. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks pp. 1–17 (mar 2016), http://arxiv.org/abs/1603.05279
20. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection (2015), http://arxiv.org/abs/1506.02640
21. Redmon, J., Farhadi, A.: YOLO9000: Better, faster, stronger. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 2017-Janua, 6517–6525 (2017)
22. Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement (apr 2018), http://arxiv.org/abs/1804.02767

23. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 39(6), 1137–1149 (jun 2015)
24. Röfer, T., Laue, T., Hasselbring, A., Heyen, J., Poppinga, B., Reichenberg, P., Roehrig, E., Thielke, F.: B-Human team report and code release 2018 (2018), only available online: `http://www.b-human.de/downloads/publications/2018/coderelease2018.pdf`
25. Speck, D., Barros, P., Weber, C., Wermter, S.: Ball localization for robocup soccer using convolutional neural networks. In: RoboCup 2016: Robot World Cup XX. Lecture Notes in Artificial Intelligence, vol. 9776, pp. 19 – 30. Springer (2017)
26. Thielke, F., Hasselbring, A.: A JIT compiler for neural network inference. In: RoboCup 2019: Robot World Cup XXIII. Springer (to appear)
27. Zickler, S., Laue, T., Birbach, O., Wongphati, M., Veloso, M.: SSL-vision: The shared vision system for the RoboCup Small Size League. In: RoboCup 2009: Robot Soccer World Cup XIII. Lecture Notes in Artificial Intelligence, vol. 5949, pp. 425–436. Springer (2010)