# Project Report: Object Detection in the RoboCup Standard Platform League

*Jakob Hartmann*

**Abstract**

This project used a state-of-the-art object detection framework to detect robots, goals, and balls in the RoboCup Standard Platform League. First a dataset was created by manually collecting images in the SimSpark simulator, which were then labeled and used for training the detector. The results on the test images and videos are very promising and allow future applications to build on them.

## Training

For this project, the object detection framework YOLO (You Only Look Once) [1] was used. The basic idea is that a neural network receives an image or video (as a series of images) as input and then detects the depicted objects and marks them with bounding boxes. This can be useful for tasks such as localizing the robot on the playing field or moving to the ball.

To train the neural network, a large amount of data is needed. Since there exists no dataset for this task and currently it is not possible to access the real NAO robot, images were taken manually using the SimSpark simulator. In total 500 images were collected for the three object classes robot, goal, and ball. 400 of them serve as training data and the remaining 100 images as validation data to evaluate the model performance and prevent overfitting. Care was taken to ensure that the images represent diverse situations to obtain realistic results. Therefore, the number of objects in the images was varied, as well as their size, the camera angle and, in the case of the robots, the pose. The images can be found in the folder *data*. Then, using the tool labelImg [2], the objects in the images were marked with bounding boxes and labeled with their corresponding classes. The tool then creates a separate text file for each image that contains the class number and the position of the bounding box for each object. Figure 1 shows an example of an image being labeled and the Figure 2 shows the corresponding text file.

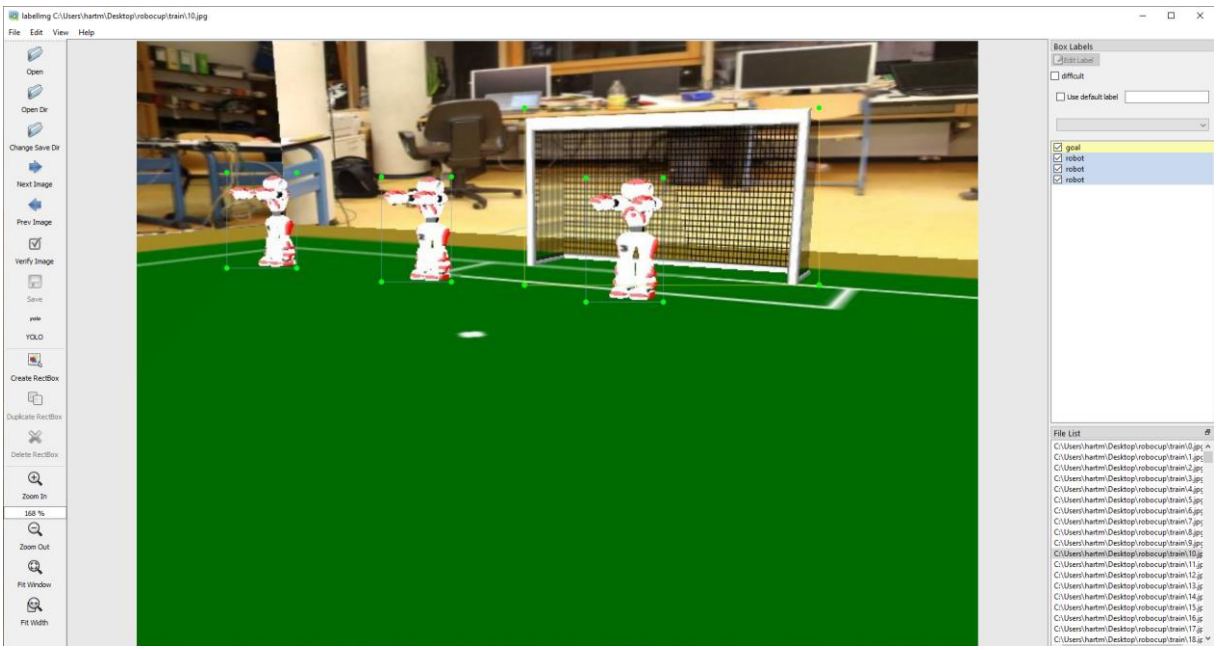


*Figure 1: Example of labeling process*

```
1 0.636650 0.254766 0.351385 0.291161
0 0.148615 0.294627 0.083123 0.155979
0 0.581234 0.324957 0.091940 0.202773
0 0.332494 0.307626 0.083123 0.171577
```

*Figure 2: YOLO format for saving the object classes and positions of the bounding boxes*

Afterwards, the darknet repository of Aleksey Bochikovskiy [3], which provides the YOLO framework, and the instructions [4] were used to train the object detector. Before the training could start, several necessary files had to be created and pre-trained weights were downloaded to speed up the training and improve the model performance. Two models were trained: the classical "YOLOv4" and "YOLOv4 tiny", a smaller version with fewer layers. Training was done using Google Colab notebooks in the cloud to speed up the learning process. After about 8 hours for YOLOv4 and 2 hours for YOLOv4 tiny, the training was completed. The resulting weights of the neural networks, as well as all other files used for training and testing, can be found in folders *yolov4* and *yolov4_tiny*.

Subsequently, the loss and mAP (mean Average Precision) of the models on the validation data was analyzed. Figures 3 and 4 show, that the loss decreases rapidly, and the mAP increases sharply during training for both models. After comparing the exact mAP values, the weights after 2.000 iterations were selected for the predictions for both models, since no substantial improvement are visible after that and to avoid the risk of overfitting.

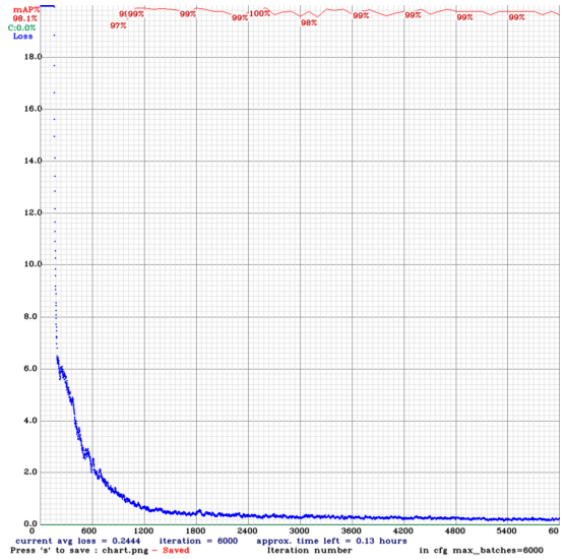


*Figure 3: Loss and mAP on the validation data during training of YOLOv4*

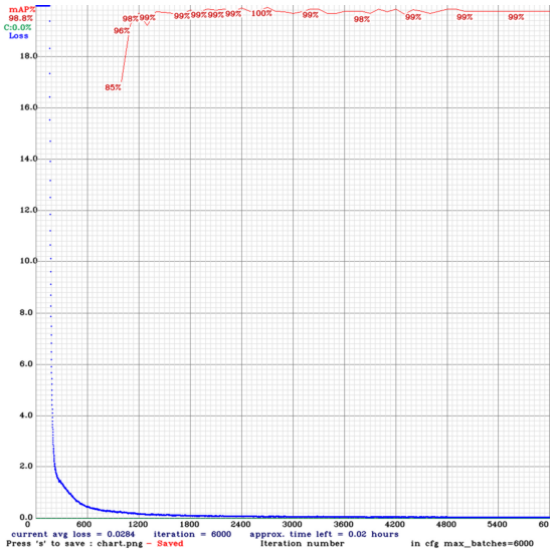


*Figure 4: Loss and mAP on the validation data during training of YOLOv4 tiny*

**Results**

Image Predictions

Both models achieve a mAP of about 99%, this suggests that we got two great object detectors! But what do these results mean in practice? To answer this question, I created a test dataset of 40 images that, like the training data, cover many different situations. For the predictions of the YOLOv4 model, a threshold of 0.9 was used, which means that only objects that the detector identified with more than 90 percent confidence are displayed. The results are excellent and can be found in the folder *image_predictions/yolov4*. In general, all objects are reliably detected, regardless of their size,

orientation, or their environment. Figure 5 shows an example of an image with several predicted objects. One problem occasionally occurs, when two robots strongly overlap, then in a few cases the hidden robot is not recognized at all, or which happens more often, both robots are recognized as one large robot, this can be seen for example in Figure 6. Here the model performance could be improved by providing more training images of such situations.

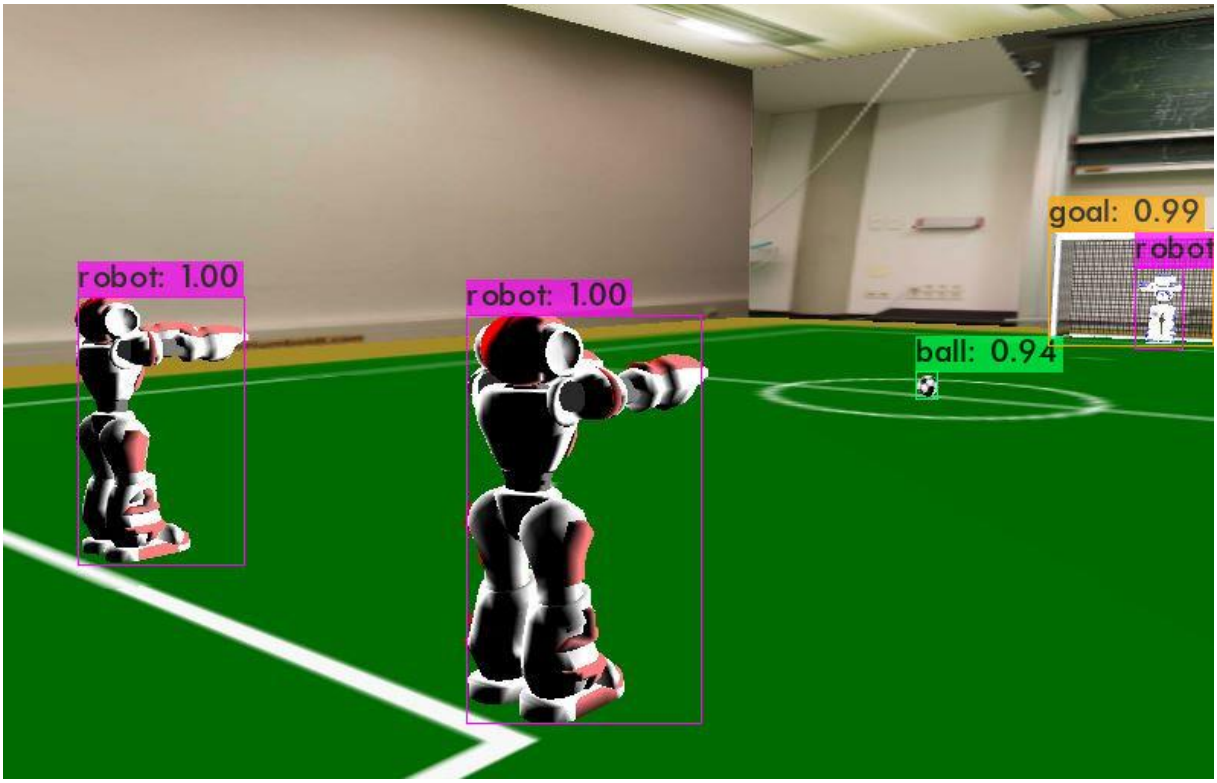


*Figure 5: Accurate detection of all objects in the image by the YOLOv4 model*

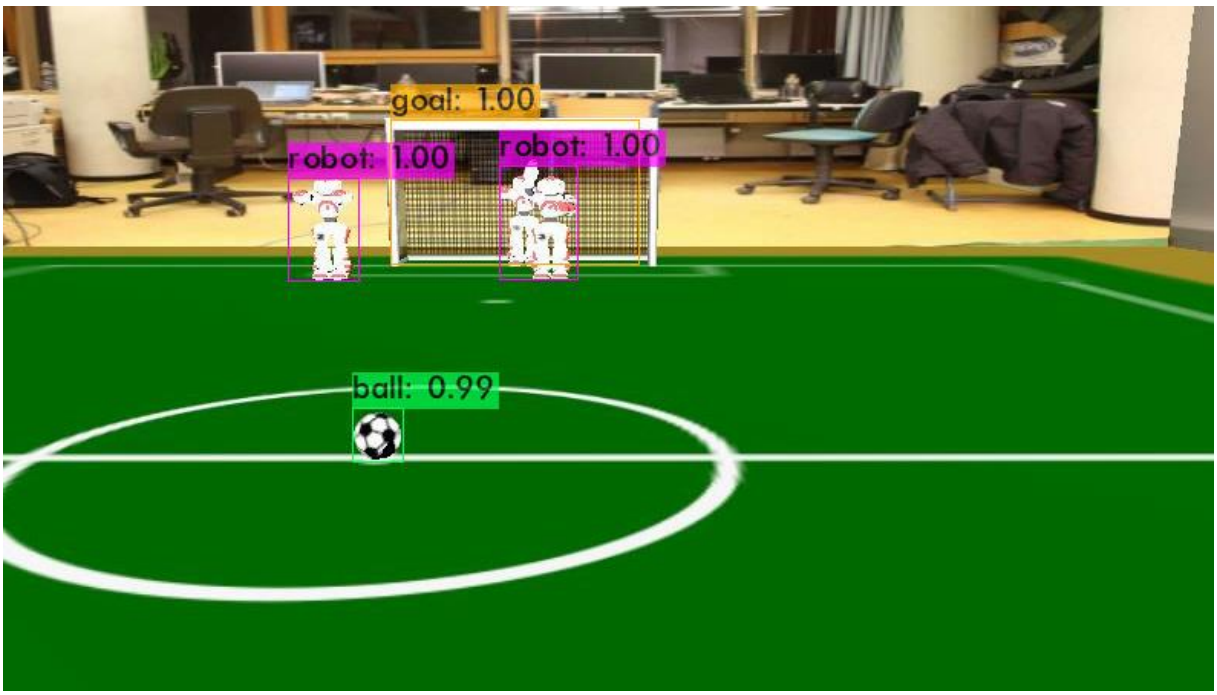


*Figure 6: Two overlapping robots are classified as one*

Especially, for the detection of robots and balls, YOLOv4 could be very useful, because the bounding boxes are set very accurately and tightly. The bounding boxes for the goals are not always as precise, here it could be helpful to only detect the goal posts or to analyze the identified image sections with goals in more detail in a future step.

However, one important question that remains for practical usability is that of speed. How fast are the objects in an image detected? For the YOLOv4 model the detection takes about 20-32 milliseconds, which is fast enough for real-time applications with 30 frames per second. BUT this speed was achieved in the Google Cloud with strong GPUs, the hardware of the NAO robot is significantly worse and slower. Without a GPU the object detection with YOLOv4 for a single image takes about 8 seconds, which is way too slow for a competition like the RoboCup Standard Platform League (SPL).

That´s why I also trained the smaller version YOLOv4 tiny to see if it could be suitable for our use case. The predictions were made with a threshold of 0.8 and can be found in the folder *image_predictions/yolov4_tiny*. They are good; however, the accuracy is worse than for the larger model. It happens more often that robots, especially if they are only partially visible, are not recognized correctly. Lowering the threshold could help to reduce this problem. But the more important question is: How fast is this model? A prediction with GPU takes about 3-5 milliseconds, with a CPU about 750 milliseconds. This is up to 10 times faster than YOLOv4, but probably still not fast enough for real-time usability when no GPU is available. However, as the hardware of the NAO robot gets improved and the neural networks become more powerful, it is still worth analyzing these models further.

Video Predictions

Therefore, I also tested the two models on videos. In the first step I recorded different scenes in the SimSpark simulator and combined them to a video. This original video, together with the predictions (threshold 0.9) can be found in folder *video_predictions*. The video now makes it possible to test the detection quality when objects are moving. For this purpose, I let the robots execute motions and moved with the camera through SimSpark to simulate a robot´s walk. Unfortunately, the camera movements are quite jerky and much faster than the walking speed of a robot. Because of this, the objects and predictions are sometimes shifted, especially when the moving camera is close to an object. However, if the camera is fixed and only the robot is moving, the object detection works very well for YOLOv4, the predictions of YOLOv4 tiny are a bit less precise, but still good. This shows that, apart from the prediction speed, the models are well suited for detecting objects in the RoboCup SPL.

Out of curiosity, I tested the models on two more videos. The first video shows the second half of a match between “UTAustinVilla” and “magmaOffenburg” in the final of the RoboCup 3D simulation tournament in 2018. The original video was obtained from [5], the predictions can be found in the folder *video_predictions/simulation_game*. Although this game also takes place in simulation, there are big differences between the video of the match and the training data taken in the SimSpark simulator. The camera angle from which the game is recorded is different, as is the turf color, the goals, and the background. Furthermore, objects such as the score, information about fouls and the players’ position on the field are displayed, which are not present in the training data. Nevertheless, YOLOv4 achieves good results with a threshold of 0.85. The robots are detected correctly in most cases, the goals in part, the ball however is not detected at all, probably due to the small size at this camera angle. That the model performs so well despite the many differences speaks for the underlying approach and the generalizability of the model. YOLOv4 tiny on the other hand reaches its limits on this task and can only detect a few objects correctly at a threshold of 0.7. For the detection to work well, the training data would have to better match the test data, especially negative samples of objects such as the score should be included to avoid false classifications.

I went one step further and tested the models, using a video showing clips from a real match between "B-Human" and "Nao-Team HTWK" in the 2018 RoboCup SPL final. The original video was obtained from [6], the predictions can be found in the folder *video_predictions/real_game*. Despite the large differences between simulation and real world, YOLOv4 produces very good results with a threshold of 0.9. The robots are detected correctly most of the time, the goals in some cases and occasionally even the ball. Compared to the simulation game, YOLOv4 tiny also performs well. With a threshold of 0.5, all object types, not just robots, are partially detected. One problem that both models face from time to time is that humans are classified as robots. It is caused by the fact that images of humans are not part of the training data. This emphasizes the importance of including negative samples of unwanted objects in the training data. Which objects these are depends on the environment the game is played in. Overall, the good results in this test illustrate that the YOLO framework is in principle well suited for detecting objects in the RoboCup SPL.

**Conclusion**

I was able to achieve the goals I set in the project proposal. The gathering and labeling of training images was very time-consuming but worked as planned and the results on the test images and the self-recorded video in SimSpark show that the object detection of robots, goals and balls works very well. Furthermore, the predictions for the simulation game as well as the real game suggest that the models transfer well to other application domains despite the differences between training and test data. I have no doubt that YOLO can be used in the RocoCup SPL, when trained with the right data. Modifications such as changing the color of the ball or playing under natural lighting conditions then no longer require new algorithms, but rather retraining the neural networks.

One big problem that remains is that of prediction speed. Despite the incredible speed of the YOLO framework compared to other object detectors, a GPU is currently needed to make real-time predictions. YOLOv4 tiny can help here and reduce the needed time significantly, unfortunately even this 10x speedup, which comes at the cost of a lower prediction accuracy, is currently not enough for real-time usability. That being said, with the continuous advances in the field of deep learning and the improving hardware of the NAO robots, it is only a matter of time until these state-of-the-art object detectors will be used in the RoboCup SPL.

**Future Work**

There are many ways to build on this project. First and foremost, of course, training the neural network with images taken by NAO robots during real games. The more training data is available and the more situations (e.g., overlapping robots) are present in the training set, the easier it is to use simpler and faster models like YOLOv4 tiny. As already mentioned, it is important to also include negative samples in the training data to prevent falsely classifying object outside the playing field (e.g., spectators of the game).

YOLO can also output the positions/coordinates of the detected objects; this can be seen in Figure 7. Thus, if the object recognition works well, other domains such a motion, localization, and behavior can expand on it. The transformation of these coordinates into the coordinate system of the robot can help him to localize itself, move to the ball or shoot it towards the goal.

```
robot: 98%     (left_x:   40   top_y:  163   width:   57   height:   95)
robot: 96%     (left_x:   72   top_y:  171   width:   68   height:  101)
robot: 99%     (left_x:  135   top_y:  172   width:   75   height:  132)
robot: 100%    (left_x:  505   top_y:  208   width:  175   height:  359)
```

*Figure 7: Detected objects together with their corresponding coordinates*

However, the most important task for deploying state-of-the-art object detectors in the RoboCup SPL is to speed up the prediction. This can for example be achieved by simplifying and optimizing the models, as has already been tried with YOLOv4 tiny, or by adapting the existing models to the limiting hardware. Implementations and tools like OpenCV-dnn [7] or Apache TVM Open Deep Learning Compiler Stack [8] can help to optimize YOLO models for running on CPUs.

**References**


[1] Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: *YOLOv4: Optimal Speed and Accuracy of Object Detection*. (2020)

[2] Lin, T.T.: *labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images*. Github.com. https://github.com/tzutalin/labelImg (accessed Feb. 26, 2021)

[3] Bochkovskiy, A.: *darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet).* Github.com. https://github.com/AlexeyAB/darknet (accessed Feb. 26, 2021)

[4] The AI Guy: *YOLOv4 Training Tutorial*. colab.research.google.com. https://colab.research.google.com/drive/1_GdoqCJWXsChrOiY8sZMr_zbr_fH-0Fg (accessed Feb. 26, 2021)

[5] AustinVilla: *RoboCup 2018 3D Simulation Final, UT Austin Villa vs magmaOffenburg - 2nd Half*. YouTube.com. Jun. 25, 2018. https://www.youtube.com/watch?v=p7MhCvbYVqE (accessed Feb. 26, 2021)

[6] HTWK Robots: *Highlights - Robocup 2018 SPL Finale: Nao-Team HTWK vs. B-Human*. YouTube.com. Dec. 18, 2018. https://www.youtube.com/watch?v=pmFKoKtRW6s (accessed Feb. 26, 2021)

[7] OpenCV: *opencv: Open Source Computer Vision Library*. Github.com. https://github.com/opencv/opencv (accessed Feb. 26, 2021)

[8] The Apache Software Foundation: *tvm: Open deep learning compiler stack for cpu, gpu and specialized accelerators*. Github.com. https://github.com/apache/tvm (accessed Feb. 26, 2021)