# Structural Pruning for Real-Time Multi-Object Detection on NAO Robots

G. Specchi[1], V. Suriani[1][0000−0003−1199−8358], M. Brienza[2], F. Laus[2], F. Maiorana[1], A. Pennisi[2][0000−0002−9081−0765], D. Nardi[1][0000−0001−6606−200X], and D. D. Bloisi[2][0000−0003−0339−8651]

[1] Dept. of Computer, Control, and Management Engineering Sapienza University of Rome, Rome (Italy), {lastname}@diag.uniroma1.it.
[2] School of Engineering, University of Basilicata, Potenza (Italy), domenico.bloisi@unibas.it

**Abstract.** In this paper, we propose a real-time multi-class detection system for the NAO V6 robot in the context of RoboCup SPL (Standard Platform League) using state-of-the-art structural pruning techniques on neural networks derived from YOLOv7-tiny. Our approach combines structural pruning and fine-tuning, to obtain a pruned network that maintains high accuracy while reducing the number of parameters and the computational complexity of the network. The system is capable of detecting various objects, including the ball, goalposts, and other robots, using the cameras of the robot. The goal has been to guarantee high speed and accuracy trade-offs suitable for the limited computational resources of the NAO robot. Moreover, we demonstrate that our system can run in real-time on the NAO robot with a frame rate of 32 frames per second on $224 \times 224$ input images, which is sufficient for soccer competitions. Our results show that our pruned networks achieve comparable accuracy to the original network while significantly reducing the computational complexity and memory requirements. We release our annotated dataset, which consists of over 4000 images of various objects in the RoboCup SPL soccer field.

**Keywords:** Deep Neural Networks · Pruning · Embedded Systems · Robot Soccer.

## 1 Introduction

RoboCup SPL is an annual international competition where teams of humanoid robots play soccer against each other. The competition requires robots to have advanced perception, decision-making, and motor control capabilities to succeed in the fast-paced and dynamic game of soccer. One critical component of a robot's perception system is object detection, which enables the robot to identify and track relevant objects, such as the ball, teammates, and opponents.

In recent years, deep learning-based object detection has emerged as a popular approach due to its ability to achieve high accuracy and robustness in complex environments. However, deploying deep learning-based object detection on

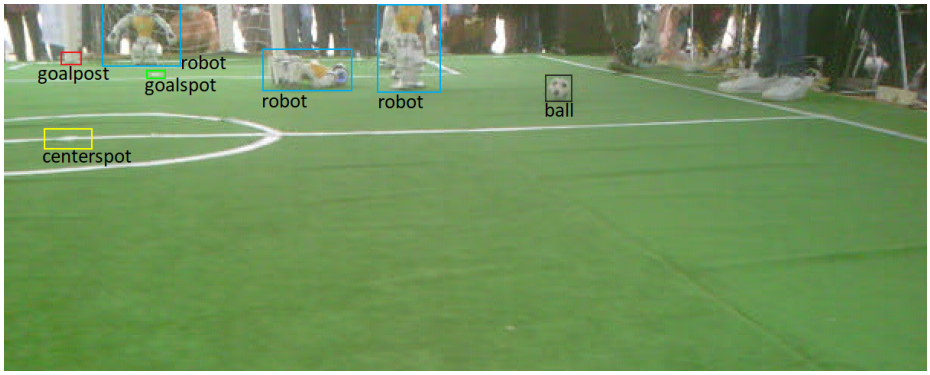



Fig. 1: An example of an annotated image with a labeled bounding box for each class: robot, center-spot, goal-spot, goal-post, and ball. The images are taken with natural light and shadows, and noisy backgrounds.

resource-constrained platforms such as humanoid robots is challenging due to their limited computational power and memory. Therefore, developing an efficient object detection pipeline that can run in real-time on these platforms is crucial. In this paper, we present a study on building a Deep Vision pipeline for the RoboCup SPL using end-to-end trainable object detection. Specifically, we focus on ball, robots, goal posts, and landmark detection, which are critical for a robot to perceive the game state and make decisions accordingly. We explored two popular detection architectures, YOLO (You Look Only Once) and SSD (Single Shot Detector). From those, we created derived models using commercial tools and structural pruning techniques in order to achieve accuracy comparable to state-of-the-art approaches but with improvement in the execution time enabling the deployment of such technologies on the NAO architecture. We created a dataset from logs collected by the SPQR team during official and unofficial events, that is publicly released. We demonstrated the effectiveness of our approach by developing a practical, multi-class object detector that can run at 32 frames per second on $224 \times 224$ input images while maintaining good detection performance on the Aldebaran NAO V6 robots.

The contribution of this work is three-fold.

1. A methodology for SSD neural network structural pruning that can be applied iteratively to obtain a greatly reduced model.
2. A quantitative comparison of different SSD approaches on NAO robots for detecting four different object classes.
3. A novel publicly available annotated dataset containing five different object classes.

The code and the additional material mentioned in this paper can be found on the following webpage: `https://sites.google.com/diag.uniroma1.it/spqr-multi-object-ssd-pruning`.

The rest of the paper is organized as follows. Section 2 contains a brief survey of the state-of-the-art and compares it to our approach; in Section 3, we expose a brief background to fully understand the concepts expressed in the paper; in Section 5 we show more in detail the proposed method; in Section 6 we discuss the experimental results obtained comparing different network models sizes inputs; and, finally, in Section 7 we discuss the conclusion obtained and the possible future developments of this work.

## 2 Related Work

Playing soccer with limited hardware in Standard Platform League has always represented a challenge also in the computer vision pipeline that each robot has to carry out onboard. With the adoption of more challenging field elements, for example, the black and white ball, the use of classic computer vision techniques has shown its limitations. The introduction of deep learning frameworks helped in migrating the classic computer vision pipelines to deep learning architectures. One of the first attempts in deploying neural networks on the NAO robot has been carried out in [1] with a two-stage approach to reduce the search space and then, a validation phase based on Convolutional Neural Networks, for multi-class classification.

To reduce the computational requirements several approaches have been proposed in literature with classic computer vision and neural network approaches. For example, several teams have explored the use of grayscale images to speed up their detectors: in [2], the team used a classical computer vision pipeline based on HAAR classifier to detect the ball on OpenCV taking advantage of the TBB library. The same idea is exploited in [4] where a whole vision pipeline for RoboCup SPL is presented that takes as input only a grayscale image to reduce the computational load. These findings suggest that grayscale images may be a promising approach for improving the efficiency of ball detection algorithms in the RoboCup SPL.

Existing deep learning frameworks usually target GPU-equipped devices and multi-core CPUs but do not perform very well on low-end hardware. Since the NAO V5 version, several approaches tackled the speed-up of the network inference on the NAO CPU. Among those, some of them have been presented to work efficiently on CPU, such as RoboDNN and the JIT Compiler [7] from the B-Human team. The former presents an approach based on the Darknet library, behaving like interpreters of neural networks built in Tensorflow and taking advantage of XLA instructions. To improve the performance, the B-Human team presents a C++ library that compiles neural network models into machine code that performs inference, targeting the Intel Atom processor of the NAO robots and outperforming the existing approaches. Both approaches are not able to manage all the layers and activations of the commercial neural networks.

Pruning techniques have been adopted in order to work efficiently with ready-to-use neural networks. In fact, in [6], the ROBO architecture uses the *weighted pruning* technique, starting from the tiny YOLO model. In the Hu-

manoid League, a YOLO architecture is used as a base for a multi-class real-time detector. In SPL, [8] presents a wide evaluation of latency and mean Average Precision (mAP) of several model architectures from the YOLO and SSD families on a range of devices representative of robot and edge device capabilities. Recently, in [5] the authors present an object detector that is robust to lighting conditions, and is able to work at 35 Hz for both cameras. This performance usually does not scale very well when passing from single to multi-class detectors and our goal was to preserve the performance of the ball perceptor, guaranteeing comparability with the single-class detector.

A more efficient way of pruning is represented by structural pruning. In [3], a general and automatic procedure for pruning is presented, which explicitly models the dependency between layers and comprehensively groups coupled parameters for pruning.

## 3 Pruning

Pruning is a technique used in neural networks to reduce the size of a trained model by removing unnecessary connections or weights. The goal of pruning is to improve the efficiency of the model by reducing its complexity without sacrificing performance. The pruning technique involves iteratively removing the connections or weights with the least impact on the model's output. The process of pruning can be done in different ways:

- *Weight pruning*: this method involves removing small weights or connections that are deemed unimportant. Weights with low absolute values are typically considered insignificant and can be removed.
- *Neuron pruning*: this method involves removing entire neurons from the network. Neurons with little or no impact on the model's output can be pruned.
- *Filter pruning*: this method involves removing entire filters or feature maps from a convolutional neural network. Filters that have little or no effect on the output can be removed.
- *Structural pruning*: removing entire sets of weights, neurons, or filters based on their structure in the network.

Pruning can be done at different stages of the neural network training, such as during initialization, after the first few epochs of training, or after the network has been fully trained. The most common approach is to prune the network after it has been fully trained. This allows for a more fine-grained evaluation of the model's performance. One popular approach is to use a technique called "magnitude-based pruning", which involves ranking the weights or activations by the magnitude and then removing the smallest ones.

Overall, pruning can help reduce the size of a neural network, improve its computational efficiency, and reduce the risk of overfitting.

### 3.1 Structural Pruning

Structural pruning is a type of neural network pruning that involves removing entire channels, filters, or layers from the neural network to reduce its computational and memory requirements. Unlike weight pruning, which sets individual weights to zero based on their magnitude, structural pruning targets entire structural components of the network. It can achieve higher levels of sparsity than weight pruning, as it can remove entire channels or filters that are redundant or not contributing significantly to the network's performance. However, structural pruning requires careful design and analysis to ensure that the pruned network maintains the necessary representational capacity to perform the desired task with high accuracy. In order to preserve as much as possible the accuracy of the unpruned network, we used an automatic pruning routine.

We adopted structural pruning as presented in [3] in order to reduce the memory requirements and have faster inference time on the NAO robot hardware.

The main concept behind structural pruning is the dependency between neurons. More specifically, if a neuron of a fully-connected neural network must be pruned, then the row of the previous weight matrix and the column of the next weight matrix corresponding to that neuron will be removed. This concept is easy to be analyzed and implemented for such a simple structure built by only fully-connected layers, but it can be complex in case of residual, concatenation, and reduction dependencies that are indeed in the YoloV7 structure. In order to keep track of the dependency of each neuron with the other neurons in the model, the approach in [3], builds a dependency graph. This graph represents the dependencies in the network and allows the execution of the pruning of any architecture in a fully automatic manner.

## 4 SPQR Dataset

The SPQR dataset consists of images acquired during friendly matches at Maker Faire 2022 in an outdoor scenario, and it includes images both from the lower and upper camera of the NAO robot. The dataset is used for training networks that have to detect robots, balls, goalposts (representing the goalposts), goal-spots (representing the penalty spot), and center-spots (representing the center circle). At the SPL Open Research Challenge at RoboCup 2022, MARIO was presented. It is a modular and extensible architecture for computing visual statistics. Two different datasets have been used, respectively one for robot and ball detection for statistics processing, and one for identifying robot poses.

The dataset for the detection task consists of 35000 images divided into 24000 for training, 3000 for validation, and 8000 for testing, and used for detecting the robots and ball. The second dataset, used for pose estimation, was created using the COCO Annotator tool, a web-based image annotation tool designed for efficiently labeling images for image localization and object detection. All annotations share the same basic data structure. The pose consists of up to

18 key points: ears, eyes, nose, neck, shoulders, elbows, wrists, hips, knees, and ankles. The annotations are stored using a JSON structure. It can be found at the following link `https://shorturl.at/uBKS3`.

To evaluate the presented approach, we chose a standard SPL dataset[8] released by the RoboEireann team, which ensures comparison with techniques developed in other works. Specifically, the dataset includes four classes, namely: robot, ball, goal-spot, and goal-post, and it has been divided into 3924 images for training, 491 images for validation, and 491 images for testing.

Fig. 2: Examples of labeled images taken from SPQR Dataset. The images also include samples with natural light conditions and noisy backgrounds, which make the detection more challenging for lightweight neural networks.

The novel dataset we released includes robots with the jersey colors yellow and red. It can be found at the following link `https://shorturl.at/biJK3`. It consists of more than 4000 images including an extra class with respect to the dataset in [8], namely the center-spot, representing the center circle. Some images are shown in Figure 3.

## 5 Proposed Approach

In RoboCup scenarios and on-edge devices, it is often hard to reach a trade-off between accuracy and performance in execution time. Often the approaches rely on the manual adaptation of the neural network by changing layers and activation functions. This becomes harder when the pruning affects a high number of network parameters.

To tackle this, we proposed a pipeline to shrink a Pytorch Neural Network in several steps, relying on an automated approach to structural pruning:

1. We start with a neural network architecture derived from YOLOv7-tiny, a popular object detection model. This network has a small footprint but it is still too heavy for real-time execution on the NAO hardware. We train this network on the RoboEireann dataset, which consists of images and annotations of soccer fields and robots used in SPL RoboCup. We manipulate the dataset with data augmentation, using the following techniques: mosaic, flip left-right, scaling, and HSV (saturation, value, and hue).
2. We apply structural pruning to the trained network to reduce its size while preserving its accuracy. Our pruning algorithm works by iteratively ranking the channels/filters according to their importance and removing the least important ones. We use a combination of L2 regularization and sensitivity-based ranking to ensure that the pruned network maintains its performance.
3. After pruning, we fine-tune the pruned network on the RoboEireann dataset to recover any loss in accuracy caused by pruning. We repeat this process of pruning and fine-tuning until we reach a desired network size that is small enough to run in real-time on a NAO robot, but still preserving accuracy in a multi-class fashion. The results of this procedure are highlighted in Table 1 and Table 2.

Combining structural pruning and fine-tuning, it has been possible to obtain a pruned network that maintains high accuracy while reducing the number of parameters and the computational complexity of the network.

To evaluate the effectiveness of our approach, we compare the performance of our pruned network with an unpruned network and with the iteratively pruned networks. The iterative pruning process consists of many iterations where fine-tune steps are followed by pruning procedure. The process stops when reaching the best trade-off between computational time and mAP.

As can be seen, our approach offers a promising solution to designing efficient neural networks for robotics applications. By using structural pruning, we are able to reduce the network size while maintaining accuracy, which is critical for real-time applications like SPL RoboCup.

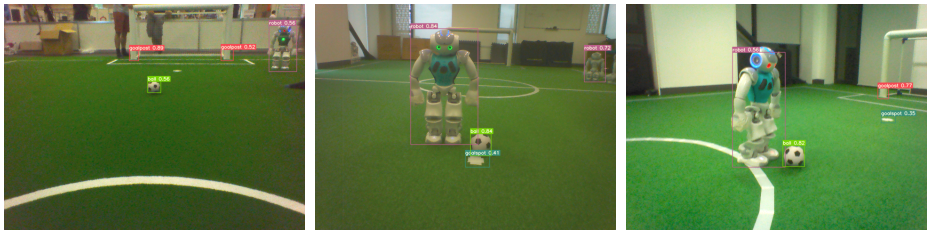
### 5.1 Experimental setup

**Software Environments** In order to deploy efficiently the Network on the robot, we performed several experiments using combinations of deep learning frameworks and optimization tools.

Firstly, we used the YOLOv7-tiny object detection network as a base network for our experiments. YOLOv7-tiny is a variant of the YOLO object detection algorithm, which uses a single neural network to simultaneously predict object classes and bounding boxes.

We performed preliminary experiments by running our PyTorch models using the Python 3.8 interpreter. Then, in order to optimize our networks, we used the ONNX (Open Neural Network Exchange) [3] framework. ONNX is an open-source framework that allows for the interoperability of deep learning models

[3] `https://github.com/onnx/onnx`



(a) (b) (c)

Fig. 3: Three examples of inference performed on images from the RoboEireann dataset. The model used for the predictions is YTv7_47K, which is a pruned version of YTv7_177K.

between different frameworks. We used ONNX to convert our PyTorch models to the ONNX format, which allows for efficient inference on different hardware platforms.

Finally, we used the OpenVINO (Open Visual Inference and Neural Network Optimization) toolkit[4] to optimize our networks for inference on Intel CPUs. OpenVINO is an open-source toolkit that provides a set of pre-trained models, optimized libraries, and tools to accelerate deep learning inference on Intel hardware. We used OpenVINO to optimize our ONNX models for inference on the NAO V6 robot.

**Used Platforms** The images in the used datasets and in the released one are acquired by the NAO V6 cameras and are saved in two different resolutions: $640 \times 480$ for the upper camera and $320 \times 240$ for the lower. For inference, the onboard computation unit of the NAO is represented by the quadcore Intel Atom E3845 CPU with 4 GB of RAM. The NAO CPU is also provided with an integrated GPU, but, as demonstrated in [5], it cannot perform better than the CPU on inference tasks.

## 6 Experimental Evaluation

To evaluate the effectiveness of our proposed approach, we conducted experiments on the NAO V6 robot equipped with the Atom E3845 CPU. We also tested networks with different input sizes and parameter sizes, ranging from 37M parameters to 7K parameters.

In fact, we trained and tested several models, as reported in the tables below. Only the model Yv7_37M is derived from YOLOv7. The model YTv7_177K is derived from YOLOv7-tiny (here denoted as YTv7_7M), that originally has 263 layers and 7M params. The name of each model represents the number of parameters of that model. The model YTv7_177K contains 177K parameters

[4] `https://docs.openvino.ai/latest/home.html`

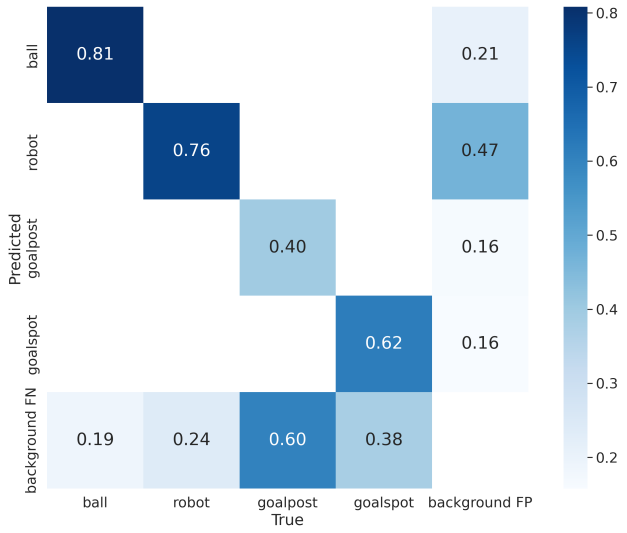



Fig. 4: Confusion matrix for the multi-class detector YTv7_47K on images with size of $224 \times 224$ pixels

and 140 layers. The model YTv7_47K is a pruned version obtained by removing half of the channels from the model YTv7_177K. The pruned model exhibited initially too low performance, due to a drop in the mAP. Then, we fine-tuned it for 371 epochs on the same training set used for training before the pruning, and it reached the performance shown in Table 1 and Table 2. The approach was applied for three iterations by pruning and then fine-tuning, obtaining the model YTv7_12K from the already pruned YTv7_47K, and we obtained YTv7_7K by applying the same approach to the model YTv7_12K.

Table 1 and Table 2 show the results of the experiments taking into account the different network and input sizes.

We report the mAP and execution time in milliseconds per image on the test set of the chosen dataset. As expected, the performance of the network improves as the input size and parameter size increase. For instance, the network YTv7_47K with an input size of $224 \times 224$ achieves the mAP of **72.1** on the NAO V6 platform.

Table 1 shows the results of the experiments for the unpruned and pruned networks executed on the NAO V6. We report the mAP and execution time for the pruned networks compared to the unpruned network. As expected, the pruned network maintains a high accuracy by reducing the size of the network. It is important to notice how the fine-tuning and pruning cycle allows us to reduce the loss of mAP while reducing the number of parameters.

Figure 4 depicts the confusion matrix for the multi-class detector YTv7_47K. We can notice that the less confused classes are ball and robot. This is due to

| Model | mAP@.5(%) - 224 × 224 | | | | |
|---|---|---|---|---|---|
| | **All** | **Ball** | **Robot** | **Goalpost** | **Goalspot** |
| Yv7_37M | 96.7 | 99.5 | 98.8 | 96.5 | 92.0 |
| YTv7_7M | 96.3 | 98.6 | 99.0 | 96.6 | 90.8 |
| YTv7_177K | 85.1 | 92.5 | 93.6 | 76.9 | 77.6 |
| **YTv7_47K** | **72.1** | 81.0 | 75.7 | 40.6 | 61.6 |
| **YTv7_12K** | 50.5 | 78.2 | 66.5 | 17.2 | 40.2 |
| **YTv7_7K** | 30.0 | 65.7 | 46.3 | 0.00 | 0.08 |

Table 1: Mean average precision computed on the test set. The input images have been scaled to 224 × 224 pixels. Models in bold represent that the model is a pruned version of the one in the upper row.

the fact that they have particular patterns easy to learn. On the other hand, the goalpost and the goal-spot can be confused very easily with other objects in the scene, because of their poor patterns. The goal-spot is easier to be recognized when close, but as soon as the robot gets far from that, the perspective makes the typical cross thin and the detector must take this into account, leading to the possibility to confuse other thin objects. It is worth noting that often, the mistakes of the detector are performed on patches that are outside the field, maybe because of a confusing background. This is usually solved in SPL by detecting the field borders and discarding perception out of them. Our approach provides an effective way to design compact and efficient neural networks for playing SPL RoboCup. The experiments show that the pruned network can achieve high accuracy while reducing the network size by several orders of magnitude. The best trade-off of accuracy and mAP are the models YTv7_47K with input dimension 224 × 224 pixels and YTv7_177K that for input dimension 160 × 160 pixels, that show similar results in terms of both latency and mAP. Although the performance is similar, we prefer to keep the input size as large as possible in order to

| Model | PyTorch (ms) | | | | OpenVINO (ms) | | | |
|---|---|---|---|---|---|---|---|---|
| | **320px** | **224px** | **160px** | **128px** | **320px** | **224px** | **160px** | **128px** |
| YTv7_177K | 164 | 106 | 67 | 51 | 121 | 56 | 30 | 20 |
| **YTv7_47K** | 114 | 79 | 54 | 41 | 65 | **32** | 20 | 15 |
| **YTv7_12K** | 44 | 39 | 27 | 21 | 49 | 28 | 14 | 10 |
| **YTv7_7K** | 32 | 22 | 17 | 14 | 45 | 23 | 13 | 9 |

Table 2: Average inference timing computed on the NAO CPU using PyTorch and on the test set of the RoboEireann dataset. Input images were resized to be squared. Smaller input sizes have been not considered to avoid a high drop in the mean average precision.

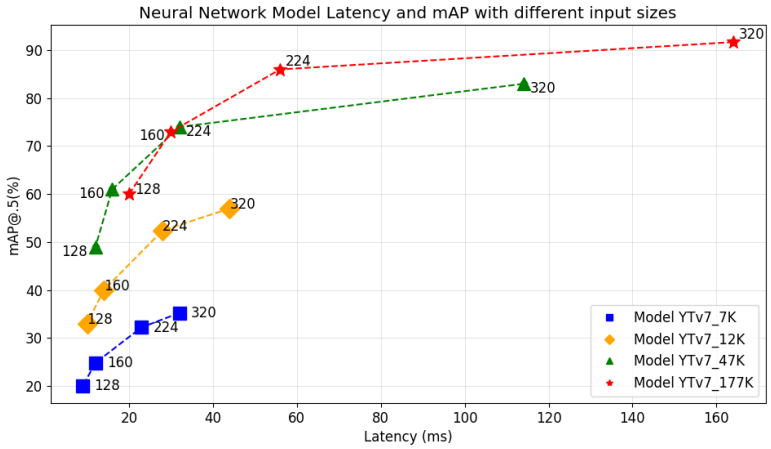



Fig. 5: Plot of relationship among latency, mAP, and input size. Each color corresponds to a specific model. For each model, we plotted four spots, one for each input size, written in the plot. Latencies were computed on the NAO V6 CPU.

have more details in the image, which are crucial when performing classification on objects that are far from the robot. Then the model we present is YTv7_47K with input dimension $224 \times 224$, that reaches a mAP of 81 even on the single-class ball, being comparable to the single-class detector previously presented in literature [5]. The performance of the network depends on the input size and parameter size, as well as the hardware platform used. In fact, in Figure 5 there is the plot of Latency, mAP, and input size of the models, related to four different input sizes. The most promising size in terms of accuracy is the highest one, but it is also the slowest. We can notice the smallest drop in performance is achieved when switching from an input image size of $320 \times 320$ to $224 \times 224$. This is due to the fact that for the specific purpose of the soccer robot, there exists a minimum requirement for the image in order to contain the visual information needed by the neural network.

## 7 Conclusions

In this paper, we presented a pipeline for pruning structural neural networks derived from YOLOv7-tiny to efficiently detect objects in the context of SPL RoboCup matches. Our approach combines structural pruning, fine-tuning, and iterative training to obtain a pruned network that maintains high accuracy while reducing the number of parameters and the computational complexity of the network. We conducted experiments on the NAO V6 robot, and we tested networks derived from YOLOv7-tiny with different input sizes and parameter sizes and measured their performance on a public SPL test set. The results show that the

networks pruned with this approach achieve admissible accuracy to the original networks but significantly reduce the computational complexity and memory requirements. With a model of 47K parameters, we can achieve real-time performance (more than 30Hz) and preserve a mAP@0.5(%) greater than 70. Such a performance is obtained thanks to the combination of deep learning frameworks and optimization tools, i.e. ONNX and OpenVINO, to train and optimize our neural networks for efficient inference on the NAO hardware. Our approach can be useful also for other robotics applications that require real-time object detection and tracking, as happened in the context of RoboCup SPL. We publicly released datasets, models, and code for iterative pruning.

As future work, we intend to prune skeleton detection approaches in order to run on the NAO algorithms to recognize game actions.

## References

1. Albani, D., Youssef, A., Suriani, V., Nardi, D., Bloisi, D.D.: A deep learning approach for object recognition with nao soccer robots. Lecture Notes in Computer Science (2017)
2. Bloisi, D., Duchetto, F.D., Manoni, T., Suriani, V.: Machine learning for realisticball detection in robocup spl (2017)
3. Fang, G., Ma, X., Song, M., Mi, M.B., Wang, X.: Depgraph: Towards any structural pruning (2023)
4. Leiva, F., Cruz, N., Bugueño, I., Ruiz-del Solar, J.: Playing soccer without colors in the spl: A convolutional neural network approach. In: Holz, D., Genter, K., Saad, M., von Stryk, O. (eds.) RoboCup 2018: Robot World Cup XXII. pp. 122–134. Springer International Publishing, Cham (2019)
5. Narayanaswami, S.K., Tec, M., Durugkar, I., Desai, S., Masetty, B., Narvekar, S., Stone, P.: Towards a real-time, low-resource, end-to-end object detection pipeline for robot soccer. In: RoboCup 2022: Robot World Cup XXV, pp. 62–74. Springer (2023)
6. Szemenyei, M., Estivill-Castro, V.: Robo: Robust, fully neural object detection for robot soccer. In: RoboCup 2019: Robot World Cup XXIII 23. pp. 309–322. Springer (2019)
7. Thielke, F., Hasselbring, A.: A JIT compiler for neural network inference. In: RoboCup 2019: Robot World Cup XXIII, pp. 448–456. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-35699-6$_3$6, https://doi.org/10.1007%2F978-3-030-35699-6_36
8. Yao, Z., Douglas, W., O'Keeffe, S., Villing, R.: Faster yolo-lite: Faster object detection on robot and edge devices. In: RoboCup 2021: Robot World Cup XXIV, pp. 226–237. Springer (2022)