In *RoboCup 2022:: Robot World Cup XXV*,
Bangkok, Thailand July 2022

Towards a Real-Time, Low-Resource, End-to-end Object Detection Pipeline for Robot Soccer

Sai Kiran Narayanaswami[1], Mauricio Tec[1], Ishan Durugkar[1], Siddharth Desai[1], Bharath Masetty[1], Sanmit Narvekar[1], and Peter Stone[1,2]

[1] Learning Agents Research Group,
The University of Texas at Austin
[2] Sony AI

Abstract. This work presents a study for building a Deep Vision pipeline suitable for the Robocup Standard Platform League, a humanoid robot soccer tournament. Specifically, we focus on end-to-end trainable object detection for effective perception using Aldebaran NAO v6 robots. The implementation of such a detector poses two major challenges, those of speed, and resource-effectiveness with respect to memory and computational power. We benchmark architectures using the YOLO and SSD detection paradigms, and identify variants that are able to achieve good detection performance for ball detection, while being able to perform rapid inference. To add to the training data for these networks, we also create a dataset from logs collected by the UT Austin Villa team during previous competitions, and set up an annotation pipeline for training. We utilize the above results and training pipeline to realize a practical, multi-class object detector that enables the robot's vision system to run at 35 Hz while maintaining good detection performance.

1 Introduction

Object detection [12, 31, 25] is one of the paramount challenges of computer vision and a key component of robotic perception. This paper develops an effective perception module for robot soccer under the stringent hardware constraints of the Robocup Standard Platform League (SPL) [5, 19, 2] , in which the vision system needs to identify various objects and landmarks in real time, such as the ball or other robots.

Neural network-based detectors have progressed tremendously over the past decade. However, many network architectures require computational power that limits their applicability in low-resource real-time scenarios such as the SPL [6]. On the other hand, the necessity for fast, resource-friendly solutions for mobile computing and the IoT has led to increasing attention on specialized hardware and network architectures [6, 16, 18, 10]. In this project, we aim to leverage these advances to develop an object detection pipeline suitable for use in the SPL. The implementation uses a software stack based on TensorFlow Lite[1] (TFLite).

The paper starts with an investigation of candidate object detection architectures by attending to reliable detection in the field and testing speed with the robot hardware. Next, it describes the methods used for data collection, labeling, and data augmentation. Finally, the performance of the detector is evaluated, and it is demonstrated to be effective at detecting objects under the computational constraints for real-time operation.

2 Background

This section goes over some preliminaries, a description of SPL, the hardware and compute setup, and the object detection challenge this paper seeks to solve.

2.1 Robocup Standard Platform League

The Robocup SPL is a humanoid soccer tournament in which our team competes as the *UT Austin Villa*. Soccer games are played between teams of five Aldebaran NAO v6 robots each, on a scaled version of a soccer field. The competition tests four main systems: vision, localization, motion and coordination. Localization relies on the vision results, such as by using detected markers to estimate the robot's position. Subsequently, the localization information is used to direct coordination. Thus reliable and fast object detection is important to achieve effective gameplay. No assistance from external hardware, including remote, is allowed (i.e., the robot's hardware is the Standard Platform).

2.2 Hardware setup

Camera There are two identical cameras available on the top and bottom of the NAOs (located above and below the eyes). The top camera is meant to give a full view of the field, while the bottom camera provides a close-up view of the ground immediately in front of the robot. Both cameras are, in principle, able to capture video at a resolution of 1280x960 pixels at 30 frames per second, with automatic exposure adjustment capabilities [27]. However, to save computation in various steps of the vision pipeline, and since the bottom camera does not view anything distant, we use a bottom camera resolution of 320x240 resolution. Example views are shown in Figure 2.

Compute The NAO has an Intel® Atom E3845 CPU, along with 4 GB of RAM [39]. The CPU has 4 cores running at 1.91 GHz, and supports some advanced SIMD instruction sets such as SSE4, enabling parallel computation. This CPU is optimized for low-power applications, and its performance compares to today's mobile processors.

An integrated GPU (Intel® HD Graphics for Intel Atom® Processor Z3700 Series) is present in the CPU, clocked at 542 MHz. It is capable of compute acceleration through OpenCL, enabled using a custom compiled Linux Kernel and drivers. This GPU is comparable in clock speed to today's mobile GPUs as well. However, experiments in Section 7.1 will show that it is not able to perform significantly faster than the CPU. Nevertheless, it remains viable as an additional source of compute power.

2.3 Object Detection Challenges

Object detection needs to run alongside the other components involved. To guarantee the functionality of all the components, the perception loop needs to maintain a processing rate of 30Hz, which comes to 33.3 ms. Within this time, both the top and bottom camera images need to be processed, and any other stages of the vision pipeline also need to be completed. These stringent timing requirements, in conjunction with the limited compute capabilities described above, challenge the implementation of a neural-network based object detector. Even real-time desktop applications such as YoloV3-Tiny[3] prove to be too expensive for these purposes.

3 Related Work

This section presents an overview of previous work related to various aspects of low-resource real-time object detection. Additional discussion of the merits and demerits of relevant approaches is handled later in the context of our design choices.

General Object Detection There are three main classes of methods for detection. One, a two-stage approach which proposes regions of interest, followed by the actual detection. Regions with CNN features (R-CNN) [12] and related methods such as Faster-RCNN [34], or Mask-RCNN [13] for segmentation fall into this class. Two, single stage detectors with *You Only Look Once* (YOLO) [31, 32, 7] being one of the most well-known paradigms that performs direct bounding box regression. Related approaches have also been developed based on corner point detection [20]. Three, detection at multiple resolutions and scales simultaneously, with methods like Single Shot MultiBox Detector (SSD) [25] and Feature Pyramid Networks [23], using existing architectures as backbones [25, 22].

Low-Resource Object Detection MobileNets [17, 37, 16] are a series of architectures developed for inference on mobile devices that use depthwise separable convolutional layers to achieve fast performance. Other lightweight architectures include Squeezenet [18] and NASNet [44], though these are mainly optimized for parameter count rather than speed as sought here. OFA Net [9] is an approach for training full scale networks, and subsequently deploying distilled versions with greatly reduced sizes. One class of approaches such as XNOR-Net [29] use binarized or heavily quantized versions of larger networks to reduce the computational cost for a given network size. Although these architectures are much less resource intensive than real-time detectors for desktop hardware [3], they are still significantly slower than what SPL requires. One major reason for this slowness is that these works target a large number of object classes, whereas there are many fewer classes in the Robocup setting.

Existing Approaches for Humanoid Soccer The SPL introduced the current NAO v6 robots in 2017. Until then, NAO v5 robots were used, which had an older, single core CPU that almost entirely prohibited deep learning pipelines. As a result, Deep Vision detectors in this competition are still in nascent stages of development, and even the winner of the 2018 competition did not use them [8]. UT Austin Villa has also been relying on more classical techniques such as SVM or feedforward classification on features extracted from region proposals [26]. The deep approaches that have been proposed [28, 36, 11] have largely been based on RoI proposals. To the best of our knowledge, these RoI-based approaches detect one object at a time. Cruz et al. [11] use XNOR-Nets for ball detection, while Rofer et al. [35] use an encoder-decoder architecture for ball localization on preprocessed RoI images. Poppinga and Laue [28] propose Jet-Nets based on MobileNet for robot detection.

xYOLO [6] is a lightweight detector proposed for the Robocup Humanoid League that is based on Tiny-YOLO [3]. It achieves good ball-detection and speed using XNOR operations and a reduced number of layers and filters. We note however, that this architecture was developed for a different league with differing robots and slightly more permissive compute constraints, including access to more powerful modern hardware such as the Raspberry Pi 3 [6]. A more recent work proposes YOLO-Lite [43], which uses lightweight inference libraries (as we also do here) and was tested on the NAO v6 robots. However, their proposed networks are not nearly fast enough to meet our target frame-rates, and they do not test the NAO's GPU.

4 Data Curation and Data Augmentation

We used three sources of data used to train our Deep Vision system. These datasets provide a good amount of variety in lighting conditions, decor, and clutter:

- **xYOLO:** This dataset contains 1400 images collected using a webcam in the Robocup Humanoid League for training the xYOLO network [6]. Annotations are provided for the ball and goalposts in the form of rectangular boxes, and we manually label other objects.
- **UTAV:** We created this internal dataset from our team logs collected during calibration and practice sessions in the SPL 2019 competition. There are about 1000 images from the top and bottom cameras, which were annotated manually for balls, robots and crosses. The images are recorded after the autoexposure adjustment.
- **NaoDevils:** This dataset contains 10 thousand images from the NaoDevils team's matches at SPL 2019 and the German Open 2019 [4]. The images contain segmentation labels from six classes: line, ball, robot, center circle, goal, and penalty cross. We transformed the segmentation masks to rectangular bounding boxes. Approximately 10% of the images are high-quality manually annotated images, the rest are lower-quality automatically segmented used for pre-training.

Manual annotation Manual annotation of existing and future data is a time-consuming process. To expedite this process, we utilize general purpose, full scale detectors to provide a seed set of labels, which can then be refined by human annotators. This idea is tested using Efficient-Det D7 [40], which is among the state-of-the-art detectors at present. We use a detector that is pretrained on the MS-COCO dataset [24]. Efficient-Det D7 was able to detect the ball and robots in many of the data images.

Thus, we use these annotations as a starting point in order to save time, as well as to allow us to use data not directly from the robot's camera for more robust detection. To this end, we used the CV Annotation Tool (CVAT) [38] by Intel, which provides a convenient user interface to execute detectors and correct their output, as well as enabling users to upload data in many data formats. Using CVAT, we annotated the ball, robots and the penalty crosses in the xYOLO and UTAV dataset images. Although we focus most of our study on ball detection with a single-class detection architecture, we also train and deploy a multi-class detector that also detects robots and penalty crosses.

Data Augmentation Multiple data augmentation techniques are used based on the Pytorch implementations available in the YOLOv5 codebase [42]. One kind of augmentations we use are kinematic augmentations which include rotations, translations and scaling. These serve to improve detections at image locations where objects are not frequently present in dataset images, and also in sizes that are not present frequently. Mosaic augmentation [7] is also used to combine multiple images, which is useful when multiple objects are present. The other class of augmentations is photometric in nature. We use transformations in the HSV color space in order to bring about invariance to lighting conditions, material color and texture, and camera exposure fluctuations.

5 Analyzing Computational Constraints

The computational requirements are critical considering that the amount of compute resources available is non-negotiable in the competition. This restriction has meant that a significant portion of our efforts so far have been directed towards understanding these constraints and studying what can be accomplished within them.

This section first describes the motivations behind the choice of software stack used. It then presents a benchmark study involving this stack to give an idea of the scale of the networks that can be executed in real time on the NAO.

5.1 Software Stack

There are several low-resource libraries available for performing inference on Machine Learning models, both general purpose [1, 30] as well as ones made by other teams competing in Robocup [41, 14]. Models are first trained in PyTorch with an adapted version of the YOLOv5 repository [42]. The network weights are then transformed to TensorFlow Lite (TFLite). We use TFLite [1] for several reasons:

- Advanced NN Layer support: TFLite has support for a much wider range of activation functions and layer types than the other libraries (e.g batch normalization).
- Multi-threaded operation: The special purpose libraries mentioned above do not allow for the use of more than one thread, whereas TFLite does.
- Optimizations for NN inference using the XNNPack[21] library, including the usage of advanced instruction sets (mainly SSE) as many other libraries also do.
- GPU Support: TFLite is the only lightweight library to have OpenCL support, thus enabling us to use the GPU as additional hardware.

5.2 Computational Benchmarks

We conduct a set of first principle benchmarks to study the computational cost as a function of network size:

We generate CNNs with random weights, with varying layer sizes and numbers of filters in each layer. Each convolutional layer has n filters, except for the last one, which has $2n$. They convert to a dense layer through a softmax of 10 units, representative of the number of objects we will eventually have to detect. The convolutional layers all use ReLU activations, and each is followed by a (2,2) Max Pooling layer. This template is based on an architecture that has been tested by the team B-Human [36]. Weights for the network were initialized at random (Gaussian with $\sigma = 0.01$). Four threads on the CPU were used to perform inference.

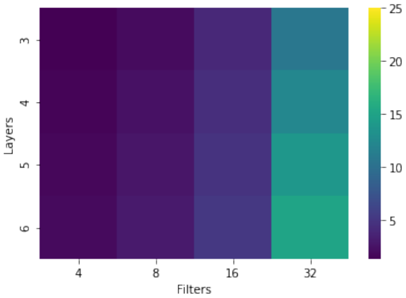


Fig. 1: Runtime benchmark results for synthetically generated CNN models. Times are presented in milliseconds.

The results are presented in Fig 1. For a target inference rate of 60 frames per second for both cameras together, we see that the network with 16 filters per layer, with 6 layers is comfortably feasible. The main impact on inference time appears to be due to the number of filters rather than the layers. This suggests that narrow, but deep architectures with skip connections might be the best computational choice.

For the above, quantization of weights is not performed. When testing with MobileNet-v2, we observed that the quantized models performed slower (64ms vs. 58ms).

6 Detector Design

As mentioned earlier, in the past, region proposal methods have been used in the SPL for detecting a single class of objects. However, this design aims to be scalable and to detect the full set of objects on the field. Region proposals tend to be more expensive with more objects to detect. Thus, this study focuses on YOLO and SSD setups.

Although SSD is more expensive than YOLO, it has potential advantages in terms of detecting objects at multiple scales. A suitable backbone architecture needs to be provided on top of which the SSD layers are implemented. We use Mobilenet-v2 [37] due to the fast runtimes while maintaining good performance in an SSD setup. SSD requires designing prior boxes for each of the feature maps from the SSD layers. However, such a design is not much more expensive compared to the design of anchor boxes for YOLO-9000 [32], which is necessary to get good performance with YOLO setups.

xYOLO [6] uses a greatly reduced version of YOLO in terms of layers and number of filters. It gets good performance on the accompanying dataset while maintaining 10 Hz detection rate on a Raspberry Pi 4, which is hardware similar to the NAOs. Taking into account the simplicity of the YOLO paradigm, we consider xYOLO a good starting point for creating reduced models. xYOLO also uses XNOR layers instead of regular convolution. They are not used in this study since XNOR layers can be detrimental to accuracy [6], and the TFLite library was not able to leverage quantization for significant speed improvements on the NAO's hardware. We also incorporate the improvements up to YOLOv3 [33], such as multiple class labels and anchor boxes.

7 Experiments

We create variants of the Mobilenet-v2+SSD and xYOLO architectures, and evaluate their performance on the task of ball detection, with the goal of identifying suitable architectures that are fast, and also maintain reasonable performance. The models are trained on both the xYOLO as well as the UTAV datasets, holding out 10% of the data as validation data. Test time performance is reported on the UTAV (test) dataset alone, as these images are closer to the expected playing conditions. Performance is measured in terms of Mean Average Precision at a confidence threshold of 0.5 (mAP@0.5).

7.1 MobileNet and xYOLO Results

The following variants of the architectures are tested:

- SSD-Mobilenet-v2: This is the same architecture as in Sandler et al. [37].
- SSD-Mobilenet-v2-6: This is a reduced version of the architecture with only 6 layers total. This reduction is achieved by not repeating layers and using a stride of 2 at each layer.
- SSD-Mobilenet-v2-8n: This is also a similarly reduced version, but it has 8 layers and half as many filters.
- xYOLO: This is the architecture from the xYOLO paper [6], but as mentioned earlier, we do not use XNOR layers in place of regular convolutional ones.
- xYOLO-nano: We drop the two widest layers from xYOLO, in addition to xNOR.
- xYOLO-pico: We replace the maxpool layers in xYOLO-nano with strided convolutions. We also drop the extra layers in the detection/classification head.

We use 224x224 inputs in batches of 4 for training all the above models. Mosaic loading is used, in conjunction with standard data augmentations such as cropping and resizing. Training iterations are performed for 100 epochs of the combined datasets. Training is done using the Pytorch framework, and the saved models are exported to TFLite format for runtime benchmarks. We use 4 threads on the NAO's CPU. 150 inferences are performed to estimate the inference times for each model. We use existing codebases for both MobileNet+SSD[3] and xYOLO [42].

The results are presented in Table 1 (left). We see that the xYOLO models are all able to achieve good mAP, while the MobileNet models struggle, particularly the reduced variants. The xYOLO-pico variant appears to have the best trade-off between speed and mAP. However, the unreduced xYOLO is also able to easily meet the 16.7 ms bound. Thus, it would remain a viable option when data becomes available for a larger number of objects.

7.2 Reduced Models for Bottom Camera

While it would be feasible to use the above models identically for both the top and bottom cameras, the bottom camera needs to detect only the ball and field markers like the cross. These also always appear at a fixed scale, and so we can reduce the detector

[3] https://github.com/qfgaohao/pytorch-ssd

Architecture	mAP@0.5(%)	CPU	GPU
xYOLO (no XNOR)	**95.6**	7.7	11.0
xYOLO-Nano	95.1	7.1	10.4
xYOLO-Pico	89.5	2.7	4.7
Mobilenet-v2	78.7	58.1	88.1
Mobilenet-v2-6	46.7	6.9	7.2
Mobilenet-v2-8	30.8	**1.1**	3.4

Architecture	mAP(%)
xYOLO-femto	85.0
xYOLO-atto	75.0
xYOLO-zepto	2.5

Table 1: Left: Performance of Mobilenet and xYOLO variants with runtimes on the CPU and GPU. Runtimes are given in ms. Right: Performance of extra xYOLO variants for only the bottom camera images.

size and the number of anchor boxes greatly. The input image size can also be reduced, leading to significant computational savings. Thus, we also benchmark the following variants of xYOLO:

– xYOLO-femto: Operates on 224x224 input with 6 layers.
– xYOLO-atto: Operates on 80x80 input, with only 4 layers.
– xYOLO-zepto: Operates on 40x40, with only 3 layers.

We also pretrain these networks on the same data as above, and fine tune on only the bottom camera images. The results are presented in Table 1 (right). We see that xYOLO-atto is able to attain decent performance comparable to xYOLO-femto, despite operating on lower 80x80 input. Thus, this size range is a viable option for the bottom camera since very high accuracy is not necessary in this case. xYOLO-zepto is unable to learn due to the drastically reduced network and input size.

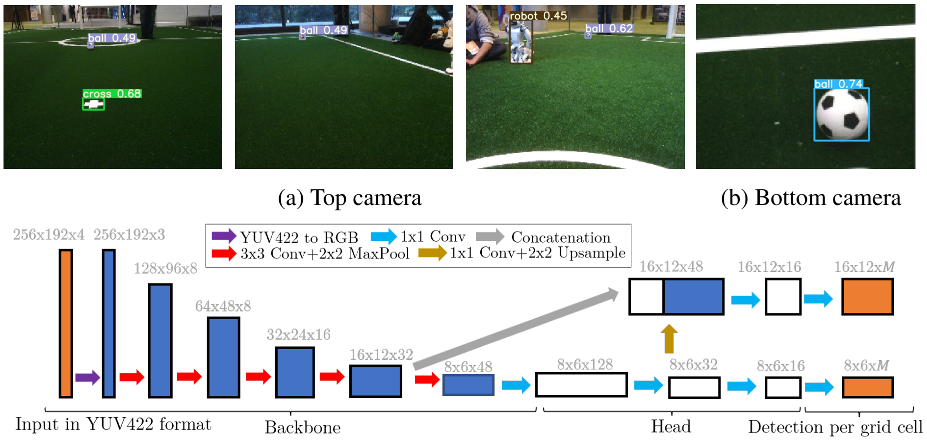
8 Practical Deployment

Although we use the above results as the basis for designing the network used in the 2021 competition, we did not base our final networks on the above architectures, and adaptations needed to be made. We describe the chosen architectures and the major adaptations in this section, followed by benchmarks and results.

Architectures We settled on particular architectures for the top and bottom camera networks as summarized in Figure 2c. The inputs are YUV-422 format images that are transformed to RGB using a (non-learnable) convolution. (See next section for additional discussion.) The bottom camera architecture is similar to the top camera but with the fist backbone layer removed. The overall architecture is based on YOLOv3 tiny [3], but aggressively shrunk in the number of channels and layers to increase speed. For the top camera, we trained and deployed the network both for single ball detection (`micro-256`) and a multi-class variant that also detects robots and crosses (`microx-256`). These two variants only differ on the last detection layer. For the bottom camera, the input size is 128x96 (`micro-128`).

Color Conversion and Subsampling The cameras are set to capture images in YUV-422 format due to the hardware characteristics favoring this format, as well as the requirements of other existing vision components. This means that the images need to be converted to RGB, the format they were trained on. We express this conversion as a pointwise convolution, and add another layer in TF-Lite to our models to perform this convolution. As mentioned earlier, the cameras capture at a higher resolution than what the model uses. As the scale factor is an integer (5), we subsample the image by dropping 4 pixels for every pixel kept. This method avoids the need to perform expensive interpolation for resizing the image.

Results Table 2 reports the individual runtimes and performances of the architectures described above. Also reported are the vision system frame-rates after the detection has been fully integrated into the vision system with the above features in place. Note that motion and localization algorithms are also running concurrently. A visualization of the results is in Figure 2. We see that the networks exhibit high detection performance and the entire vision system is able to run at at least 30 Hz for multi-class detection, which goes up to 35 Hz when the top and bottom detectors are run simultaneously in a multithreaded setup. Higher framerates enable more reliable estimates of object positions, as well as better localization.



(a) Top camera

(b) Bottom camera

(c) Architecture

Fig. 2: Detection examples for the network used in the 2021 competition. (c) shows the architecture for the top camera.

9 Conclusions

Through this project, we have gained understanding of the challenges involved in object detection for the SPL. We implemented a data collection pipeline to enable training

Network	mAP@0.5	CPU(3 threads)	GPU
micro-256	98.7	9.9	15.4
microx-256	97.2	14.6	19.1
micro-128	98.3	2.9	5.3

Top Network	Framerate
micro-256	33
microx-256	30.5
microx-256 (simultaneous)	35

Table 2: Left: Performances and runtimes (ms) of deployment architectures. Right: vision system framerates (Hz) of different combinations (bottom network is always micro-128). "Simultaneous" indicates top and bottom networks run simultaneously on CPU (3 threads) and GPU respectively. Otherwise, they are run one after the other.

our object detectors, and data augmentation strategies for robustness of detections. We identified network architectures and ways to improve on runtime while maintaining good detection performance. We put these to the test, and realize a practical, multi-class object detector that is robust to lighting conditions, and is able to work at 35 Hz for both cameras. There are several avenues to pursue in the future:

- There has been work based on Stochastic Scene Generation specific to Robosoccer [15] using the Unreal Engine to render realistic Robosoccer game scenes, which provides a promising avenue for data augmentation in our proposed training pipeline, and we are currently investigating the feasibility of using such an approach. In conjunction with GANs, such a setup could be used to perform high quality data augmentation for increased robustness.
- Depthwise Separable Convolutional are used in MobileNets, leading to significant computational savings, without degradation in detection performance. We conducted initial tests in our setting obtaining promising results. Thus, these layers open avenues for using larger networks for better performance and speed.

10 Acknowledgements

The authors thank Juhyun Lee, Terry Heo and others at Google for their invaluable help with setting up and understanding the best practices in using TensorFlow Lite. This work has taken place in the Learning Agents Research Group (LARG) at the Department of Computer Science, The University of Texas at Austin. LARG research is supported in part by the National Science Foundation (CPS-1739964, IIS-1724157, FAIN-2019844), the Office of Naval Research (N00014-18-2243), Army Research Office (W911NF-19-2-0333), DARPA, Lockheed Martin, General Motors, Bosch, and Good Systems, a research grand challenge at the University of Texas at Austin. The views and conclusions contained in this document are those of the authors alone. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

Bibliography

[1] Abadi, M. et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), software available from `https://www.tensorflow.org/`

[2] Achim, S., Stone, P., Veloso, M.: Building a dedicated robotic soccer system. In: Proceedings of the IROS-96 Workshop on RoboCup, pp. 41–48 (November 1996)

[3] Adarsh, P., Rathi, P., Kumar, M.: Yolo v3-tiny: Object detection and recognition using one stage improved model. In: 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), pp. 687–694 (2020)

[4] et al, P.B.: Robocup spl instance segmentation dataset. `https://www.kaggle.com/pietbroemmel/naodevils-segmentation-upper-camera` (2019)

[5] Alami, R., Biswas, J., Cakmak, M., Obst, O. (eds.): RoboCup 2021: Robot World Cup XXIV. Springer, Cham (2022), ISBN 978-3-030-98681-0

[6] Barry, D., Shah, M., Keijsers, M., Khan, H., Hopman, B.: xyolo: A model for real-time object detection in humanoid soccer on low-end hardware. In: 2019 International Conference on Image and Vision Computing New Zealand (IVCNZ), pp. 1–6 (2019)

[7] Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934 (2020)

[8] Brameld, K., Hamersley, F., Jones, E., Kaur, T., Li, L., Lu, W., Pagnucco, M., Sammut, C., Sheh, Q., Schmidt, P., Wiley, T., Wondo, A., Yang, K.: Robocup spl 2018 runswift team paper (2019)

[9] Cai, H., Gan, C., Han, S.: Once for all: Train one network and specialize it for efficient deployment. CoRR **abs/1908.09791** (2019)

[10] Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment (2019)

[11] 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. CoRR **abs/1706.06702** (2017)

[12] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)

[13] He, K., Gkioxari, G., Dollar, P., Girshick, R.: Mask r-cnn. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2980–2988 (2017)

[14] Hermann, T.: frugally-deep: Header-only library for using keras models in c++. Robocup (2019)

[15] Hess, T., Mundt, M., Weis, T., Ramesh, V.: Large-scale stochastic scene generation and semantic annotation for deep convolutional neural network training in the robocup spl. In: Akiyama, H., Obst, O., Sammut, C., Tonidandel, F. (eds.) RoboCup 2017: Robot World Cup XXI, pp. 33–44, Springer International Publishing, Cham (2018), ISBN 978-3-030-00308-1

[16] Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., Le, Q.: Searching for mobilenetv3. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 1314–1324 (2019)

[17] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR **abs/1704.04861** (2017)

[18] 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 (2016)

[19] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: The robot world cup initiative. In: Proceedings of the First International Conference on Autonomous Agents, pp. 340–347, AGENTS '97, Association for Computing Machinery, New York, NY, USA (1997), ISBN 0897918770, https://doi.org/10.1145/267658.267738

[20] Law, H., Deng, J.: Cornernet: Detecting objects as paired keypoints. In: Proceedings of the European Conference on Computer Vision (ECCV) (September 2018)

[21] Lee, J., Barchard, F., Guang, M., Mei, Y.C., Duke, J., Elsen, E., Dukhan, M., Kulik, A.: Xnnpack. `https://github.com/google/XNNPACK` (2019)

[22] Lin, T., Goyal, P., Girshick, R.B., He, K., Dollár, P.: Focal loss for dense object detection. CoRR **abs/1708.02002** (2017)

[23] Lin, T.Y., Dollar, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)

[24] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision – ECCV 2014, pp. 740–755, Springer International Publishing, Cham (2014), ISBN 978-3-319-10602-1

[25] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: SSD: single shot multibox detector. CoRR **abs/1512.02325** (2015)

[26] 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 Soccer World Cup XXI, pp. 45–59, Springer (July 2017)

[27] Omnivision Technologies: Ov5640: color cmos qsxga (5 megapixel) image sensor with omnibsi(tm) technology (2011)

[28] Poppinga, B., Laue, T.: Jet-net: Real-time object detection for mobile robots. In: Robot World Cup, pp. 227–240 (2019)

[29] Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. CoRR **abs/1603.05279** (2016)

[30] Redmon, J.: Darknet: Open source neural networks in c. http://pjreddie.com/darknet/ (2013–2016)

[31] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788 (2016)

[32] Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517–6525 (2017)

[33] Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. CoRR **abs/1804.02767** (2018)

[34] Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28, pp. 91–99, Curran Associates, Inc. (2015)

[35] Rofer, T., Laue, T., Baude, A.: Team report and code release 2019 (2019)

[36] Röfer, T., Laue, T., Hasselbring, A., Heyen, J., Poppinga, B., Reichenberg, P., Roehrig, E., Thielke, F.: B-Human team report and code release 2018. http://www.b-human.de/downloads/publications/2018/CodeRelease2018.pdf (2018)

[37] Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. CoRR **abs/1801.04381** (2018)

[38] Sekachev, B., Zhavoronkov, A., Manovich, N.: Computer vision annotation tool: A universal approach to data annotation. https://cvat.org (2019)

[39] Softbank Robotics: Nao 6 datasheet. https://www.generationrobots.com/media/Specifications_NAO6.pdf (2018)

[40] Tan, M., Pang, R., Le, Q.V.: Efficientdet: Scalable and efficient object detection. CoRR **abs/1911.09070** (2019)

[41] Thielke, F., Hasselbring, A.: A jit compiler for neural network inference. In: Chalup, S., Niemueller, T., Suthakorn, J., Williams, M.A. (eds.) RoboCup 2019: Robot World Cup XXIII, pp. 448–456, Springer International Publishing, Cham (2019), ISBN 978-3-030-35699-6

[42] Ultralytics: Yolov5 rocket in pytorch. https://zenodo.org/badge/latestdoi/264818686 (2020)

[43] Yao, Z., Douglas, W., O'Keeffe, S., Villing, R.: Faster yolo-lite: Faster object detection on robot and edge devices. In: Alami, R., Biswas, J., Cakmak, M., Obst, O. (eds.) RoboCup 2021: Robot World Cup XXIV, pp. 226–237, Springer International Publishing, Cham (2022), ISBN 978-3-030-98682-7

[44] Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. CoRR **abs/1707.07012** (2017)