Neural Computing and Applications
https://doi.org/10.1007/s00521-021-05972-1

S.I. : NCACVIP

Fully neural object detection solutions for robot soccer

Márton Szemenyei[1] · Vladimir Estivill-Castro[2]

Received: 16 October 2020 / Accepted: 25 March 2021

Abstract
RoboCup is one of the major global AI events, gathering hundreds of teams from the world's best universities to compete in various tasks ranging from soccer to home assistance and rescue. The commonality of these three seemingly dissimilar tasks is that in order to perform well, the robot needs to excel at the all major AI tasks: perception, control, navigation, strategy and planning. In this work, we focus on the first of these by presenting what is—to our knowledge—the first fully neural vision system for the Nao robot soccer. This is a challenging task, mainly due to the limited computational capabilities of the Nao robot. In this paper, we propose two novel neural network architectures for semantic segmentation and object detection that ensure low-cost inference, while improving accuracy by exploiting the properties of the environment. These models use synthetic transfer learning to be able to learn from a low number of hand-labeled images. The experiments show that our models outperform state-of-the-art methods such as Tiny YOLO at a fraction of the cost.

Keywords Robot soccer · Neural networks · Computer vision · Sim2real

1 Introduction

The stated goal of RoboCup Soccer is to design a team of autonomous robots that are able to defeat the world champion soccer team using FIFA rules by the middle of the twenty-first century. Achieving human-level vision and scene understanding is an essential component of achieving this goal. In accordance with this insight, the RoboCup environment has steadily changed from featuring objects that are easy to recognize using low-level features, such as color, to ones that greatly resemble the actual objects used in human soccer. The RoboCup 2018 competition held in Montreal included several leagues that jointly involved 2,350 human on-site participants from 39 countries using more than 1,000 robots. In 1995, also in Montreal, but 23 years earlier the announcement for the first RoboCup was made although the first RoboCup was held until 1997 with only three leagues but more than 40 teams. Nevertheless, the original idea for robots playing soccer is attributed to an earlier analysis of research goals in artificial intelligence architectures and computer vision challenges posed by Mackworth [1].

To achieve real-time computer vision [2] with the more challenging soccer field, the vision pipelines used by the competing teams have changed in tandem, going from human-engineered vision methods [3, 4] to pipelines relying increasingly on machine learning. In 2016, deep learning was used for validation phase after a color segmentation phase [5]. Several teams have used convolutional neural networks either for binary classification tasks [6, 7] or to detect several relevant object categories [8, 9], while detecting robots (humanoids) is also relevant [6]. Obviously, the focus is at least one object, the ball [10, 11]. Ball-only CNNs had input size massively reduced to be ported to typical robots for the humanoid league [12] or, alternatively, dedicated GPU-hardware is placed in board of the robots [13]. Later, a technique called Visual Mesh [14] was used to improve the performance of neural networks at multiple scales.

These methods, however, use CNNs for classification only; therefore they still require a separate object proposal method, and the quality of the system may largely depend

✉ Márton Szemenyei
szemenyei@iit.bme.hu

[1] Department of Control Engirneering and Information Technology, Budapest University of Technology and Economics, Magyar Tudosok krt. 2., 1117 Budapest, Hungary

[2] Department of Information and Communication Technologies, Pompeu Fabra University, Roc Boronat, 138, Barcelona 08018, Spain

Published online: 15 April 2021

on the efficiency of the algorithm used to generate candidates for classification. A further disadvantage is that running the same neural network on potentially overlapping image regions is wasteful, since the same features are computed twice. A more ambitious approach is to achieve a color-free vision system [15]; however, this work still uses an earlier phase to generate proposals to the CNNs for specific objects such as the ball and other robots. This earlier phase consists of classical pattern-recognition methods that identify regions of high contrast.

One of the most important advances of recent years is the work published by Hess et al. [16] in which they present a high-quality virtual RoboCup environment created with Unreal Engine™. This scene generation enables anyone to create large datasets of realistic images of a soccer field along with pixel-level semantic labeling. Semantic annotation was profoundly valuable for our research because the performance of a trained neural network is highly dependent on the quality and quantity of the training data. The alternative (to create a large hand-labeled database) is prohibitively highly time-consuming. Notably, Dijk and Scheunemann [17] used a deep neural network to perform semantic segmentation on limited hardware. Their solution is capable of detecting balls and goalposts at multiple resolutions in real-time.

We insist that most of these neural networks methods were in the realm of classification and needed a separate object proposal method. The approach suffers from wasted computation for overlapping proposals, and it prevents the networks from using the larger context of the object. Standard object detection or segmentation methods such as YOLO [18] or U-Net [19] provide a solution to this problem, yet they are far too computationally expensive to be feasible for the Nao robots. For example, for the Pepper robot, researchers have added an external device to efficiently run Tiny YOLO [20].

In this paper, two novel architectures are presented: ROBO-UNet for semantic segmentation and ROBO for object detection. We believe this paper is illustrative and provides strategies to consider tailor-made models. The particular case-study environment is the RoboCup competition. Nevertheless, we suggest that encoding certain properties of the setting in the architecture while keeping the number of parameters low will generally result in solutions that achieve superb accuracy at low computational costs. We also employ synthetic transfer learning to allow the models to learn from relatively small hand-labeled datasets. Transfer learning aims at reusing knowledge gained from one problem and applying it to a different but related problem [21]. For RoboCup, it has been mainly used for reinforcement learning of behaviors; in particular, playing *Keepaway* is considered a stepping-stone to learning all behaviors for the game of soccer [22, 23]. However, for CNNs, synthetic transfer learning avoids the costly exercise of capturing and labeling real images by collecting data from a simulated environment [24]. The model learned in the simulation is reused at a later refinement phase with fewer real images.

To further reduce execution times, the models are pruned during training. In order to reap the benefits of this during inference, we created a new library, called RoboDNN, which is a lightweight, inference-only library that is optimized for running pruned networks on the Nao robot. The library has no dependencies, making it easy to compile for the robots. We also use label propagation to predict future labels from previous ones to increase the speed of the pipeline even further.

We compared our proposed architectures with state-of-the-art methods, such as Tiny YOLO v3 [25] and U-Net [19]. We report the results showing superior detection accuracy at considerably lower computational costs. The datasets and code used for training, as well as the RoboDNN library are available publicly [26]. This paper expands on our previous work [27] by presenting new architectures and variants, expanded datasets and experiments and new methods to decrease run time.

2 Related work

2.1 Detection and segmentation

Object detection and segmentation are two of the fundamental tasks in computer vision with numerous applications ranging from autonomous driving [28] to medical imaging [29]. The first successful architectures for object detection were region-based models [30], which first, generated region proposals, and performed classification and bounding box regression on the proposals independently. Later, these architectures were succeeded by single-shot detectors [31, 32], which were able to achieve considerably higher speeds by doing away with region proposals entirely.

The most widely-used single shot detector is the YOLO model, which has several versions and model variants [18, 25], including models designed for high-speed inference. The most recent high-speed version of YOLO is the Tiny YOLO v3 model. Tiny YOLO is a fully convolutional network with a stride of 32, meaning that if given a standard 416×416 input image, the output activation array has spatial dimensions of 13×13. The final layer of the model is a 1×1 convolutional layer predicting B bounding boxes at every location (grid cell). Each bounding box has $5 + N_c$ parameters, which are the center coordinates, width and height of the bounding box, the confidence score, and the N_c class scores.

Tiny YOLO also uses so-called anchor boxes, reference bounding boxes, acquired by running clustering on the boxes in the training set. The width and height of the bounding boxes are predicted relatively to one of the B anchor boxes. The center coordinates are predicted relatively to the grid cell. Predicting a certain object is the responsibility of the output with the most similar anchor box at the grid cell that contains the center of the object.

The networks used for semantic segmentation use a symmetrical, fully convolutional encoder-decoder architecture, the most notable examples being FCN [33] and U-Net [19]. In recent years, numerous enhanced architectures have been proposed: By using atrous convolution [34] and pooling, the networks field of vision can be improved, while spatial pyramid pooling increases robustness to scaling [35]. Several approaches have been proposed to combine segmentation networks with Conditional Random Fields (CRF) [34, 36] to improve the models' understanding of contextual information. Notably, object detection and segmentation networks can be combined to perform instance segmentation [37].

2.2 Transfer learning

Transfer learning [38] is a common practice to reduce the need for labeled training samples and thus decrease the cost of supervised learning. In the standard scenario, the network is pre-trained on a large dataset in the source domain, then fine-tuned on a considerably smaller dataset in the target domain. The quality of the resulting neural network is largely dependent on their aspects (1) the generality of the learned features on the source dataset, (2) the similarity of the two domains, and (3) what transfer learning procedure is applied [39].

Due to its success, transfer learning is applied widely for several tasks, including object detection and semantic segmentation [40]. Interestingly, similar ideas are employed not only in supervised learning, but other areas as well. Weight agnostic networks [41], for instance, search for an architecture that performs well on a domain with randomly sampled weights. The main difference is that here, the aim is to find a model that performs well regardless of the weights.

A major use-case of transfer learning is when a large dataset of synthetic training examples is available with automatically generated labels. In this case, instead of changing the domain, we change the data source. Variational autoencoders can also be used to train a detection network with minimal real labeled examples, as shown in the domain of brain vasculature segmentation [42]. In radical contrast to the vanilla application of transfer learning which typically re-trains the last layers of a network, we incorporate synthetic transfer learning by re-training the first few layers of the network. To the best of our knowledge this is an innovative use of transfer learning. We provide a justification for this novel choice. The justification goes in hand with the previously mentioned substitution of data source.

3 Architectures

In this section, we detail the proposed architectures ROBO and ROBO-UNet. The common design principles connecting these architectures are computational efficiency and low parameter count. For this reason, our models use a relatively low number of filters, while they also tend to downscale the input image aggressively to avoid computation on high resolutions. The original Tiny YOLO v3 and UNet architectures are shown in Fig. 1a, b.

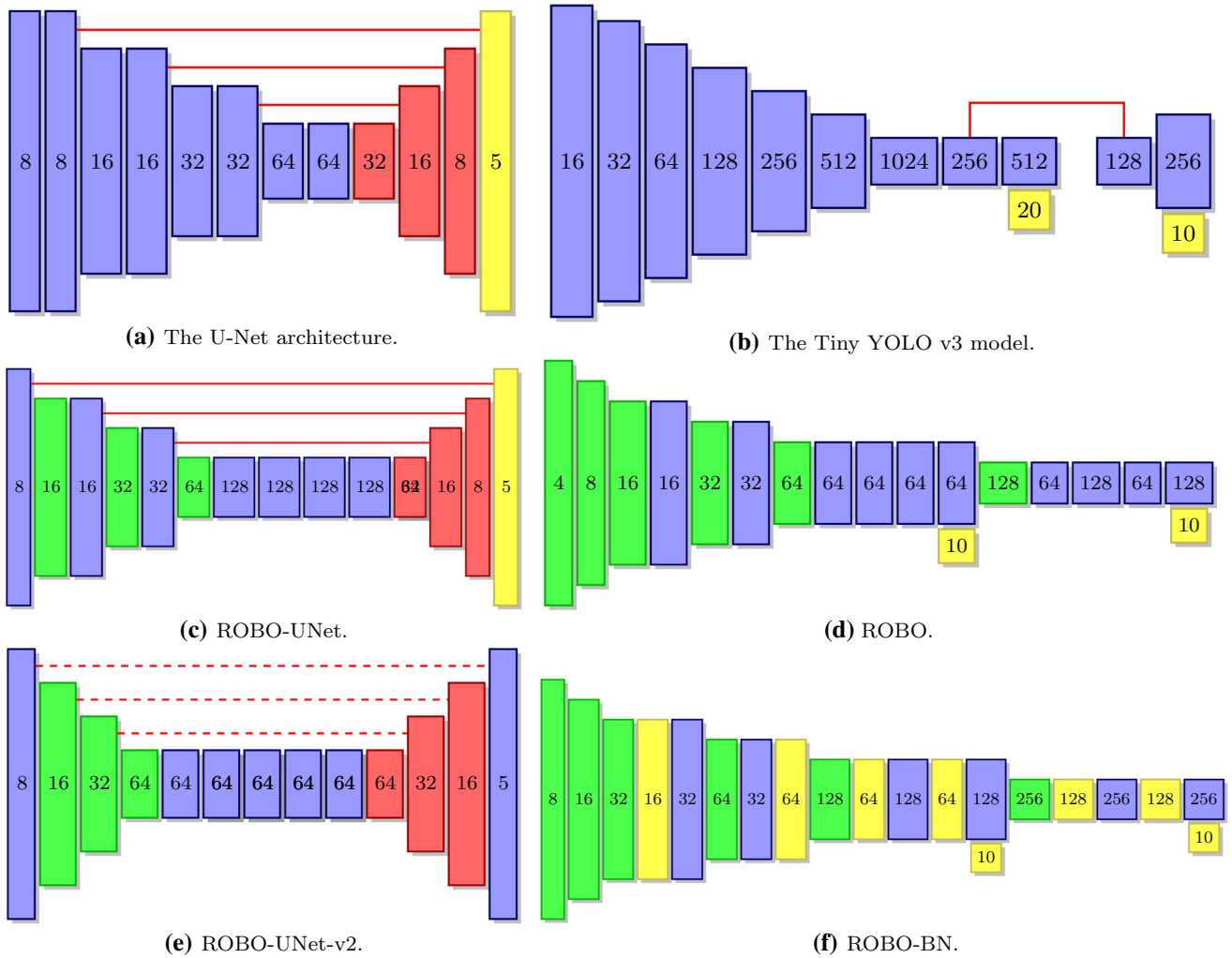
3.1 ROBO

The proposed ROBO model follows the popular Tiny YOLO [25] architecture; however, we introduce several crucial changes to the original model to optimize its performance and to fine-tune its architecture to the robot soccer setting.

3.1.1 Performance optimization

Despite its name, Tiny YOLO is a medium-sized network, with 20 layers (including pooling), some of which have 512 or even 1024 channels. This network was designed to perform well on complex datasets, such as COCO [43]. To use such a large network for object detection in the robot soccer setting is currently impractical. Therefore, we propose the ROBO architecture, which is a 16-layer, fully convolutional network with the widest convolutional layer having only 256 channels. This reduction in the number of channels is justified, since the robot soccer environment is considerably less complex and less varied than generic object recognition.

The ROBO architecture also replaces the max pooling layers in Tiny YOLO with strided convolution. This replacement is beneficial, since max pooling discards spatial information, reducing the performance of neural networks even when the spatial information is considerably less important, such as classification [44]. Arguably the effect of using max pooling is even worse for detection. To preserve more spatial information during downscaling, every strided convolutional layer increases the number of channels. Furthermore, using strided convolution for downscaling increases the complexity of the features learned by ROBO, since the network base has 15



(a) The U-Net architecture.

(b) The Tiny YOLO v3 model.

(c) ROBO-UNet.

(d) ROBO.

(e) ROBO-UNet-v2.

(f) ROBO-BN.

Fig. 1 Our architectures: the green layers correspond to strided convolution, while red nodes are transposed convolution. The blue and yellow layers are 3×3 and 1×1 convolutions, respectively. Batch normalization is included in every layer. Red lines denote skip connections with addition, while dashed red lines use concatenation

subsequent convolutions, while Tiny YOLO only has 9 and 11 for its two outputs respectively.

To further increase the model's speed, the input image is downscaled aggressively. The first three layers of the network are all strided convolution, and thus, the spatial dimensions of the feature maps are reduced eightfold by the time the first conventional convolutional layer is applied. The total stride of the network (the ratio between the spatial size of input and output) is increased from 32 to 64, making the final part of the network four times faster. Notably, the aggressive downscaling and increased stride should, in theory, decrease the network's accuracy for smaller objects, but the replacement of max pooling should counter accuracy loss in small objects.

3.1.2 Exploiting the environment

The ROBO architecture also exploits the fixed aspect ratio of the Nao robot's camera. All variants of YOLO are designed to handle images of all size and shape, requiring a complex preprocessing step, where all images are padded and resized to 416×416. We avoid completely the wasteful computation on padded parts of the image. The Nao robots have a 4:3 ratio camera, meaning that we can choose a fixed resolution of $k * 64 * (4 \times 3)$, where k is a positive integer to ensure that no pixel information or computation is wasted. In this paper, we chose $k = 2$, which results in an input resolution of 512×384. Choosing a larger grid size would result in insufficiently fast image-processing rate.

The ROBO architecture also exploits prior knowledge about the relevant objects and their arrangement to make

simplifications to the final layer of the model. The method uses the following properties of soccer fields.

- There are four classes (ball, cross, robot and goalpost).
- There is a limited number of all classes in the field.
- Objects of the same class are not cluttered (robots are mild exceptions to this rule).
- Objects of the same class have similarly shaped bounding boxes (robots are mild exceptions to this rule, since they might fall over).

For the above reason, the ROBO model uses class-specific anchor boxes, meaning that from each cell on the final 8×6 grid, it makes exactly one prediction for each class. The anchor boxes are also computed separately per class by simply averaging the bounding box widths and heights. (Our method removes the need for clustering.) Since the index of the anchor box now determines the class, the classification scores can be removed, meaning that our network has $N_{\text{class}} * 5 = 20$ outputs. This change simplifies both the loss function and the inference process since the classification loss no longer has to be calculated. Moreover, non-maximum suppression is no longer necessary during training, since it is only performed between predictions from the same grid cell that have the same class.

Since robots are mild exceptions to some of the rules above, we experimented with allocating 2 anchor boxes for the robot class, instead of just one. However, this did not yield noticeably different results. Even on a validation set containing a fair number of cluttered or fallen robots, the multi-box version was not able to detect robots more accurately. In the cluttered scenario the single-box model sometimes predicts a single bounding box that encompasses both robots. In our opinion, correcting this minor issue is not worth the added complexity. Some examples can be seen in Fig. 2.

Finally, we changed the output logic of Tiny YOLO slightly: instead of upscaling the final feature layer of the network to produce an upscaled output, we simply produce the upscaled output from an earlier layer in the network. Also, instead of predicting all four classes at both outputs, the original output is responsible for predicting robots and goalposts, while the upscaled one predicts balls and crossings only. Our primary reason for doing this is that the ball and crossing classes are usually much smaller than the other two, so predicting them from a higher resolution feature map will increase the localization accuracy considerably.

For our experiments, we also made a slightly different version of ROBO, called ROBO-Bottleneck, or ROBO-BN for short. In this architecture, we doubled the number of channels in every convolutional layer. To account for the higher number of parameters and computational cost, we added 1×1 bottleneck convolutional layers to reduce the number of channels before 3×3 convolutions. The ROBO and ROBO-BN architectures are shown in Fig. 1d, f.

3.2 ROBO-UNet

Our second proposed architecture is based on UNet [19], using max pooling for downscaling and transposed convolution for upscaling, as well as employing dilated convolutions to increase the field of view of the final classification layer. This first design has three modules consisting of convolutional and downscaling layers, combined with three upscaling layers. The proposed segmentation architectures replace pooling with strided convolution.

Most CNNs used for semantic segmentation are relatively waist-heavy, meaning that the middle section of the

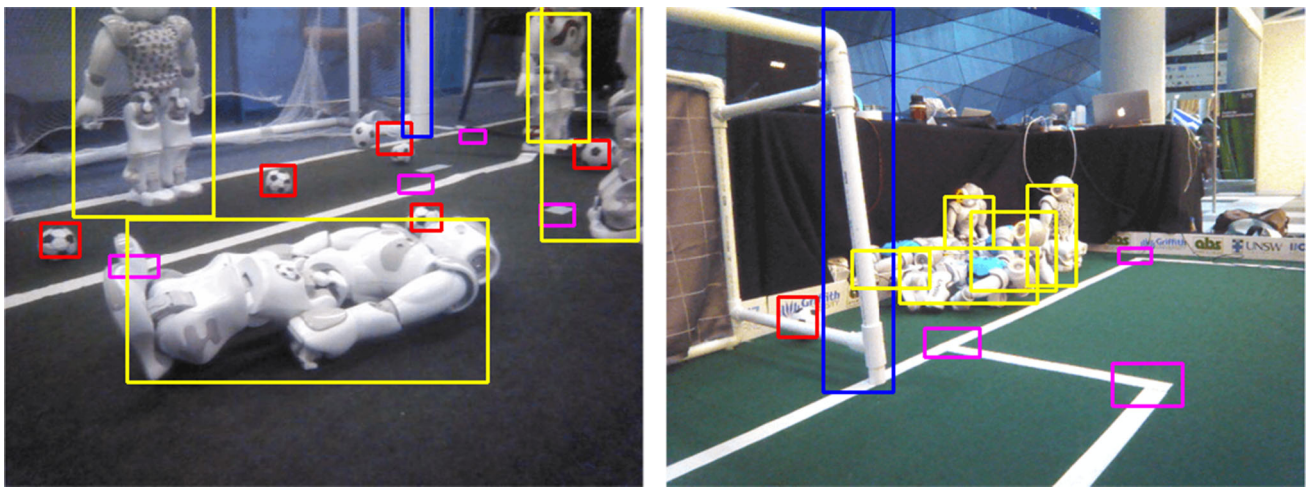


Fig. 2 Examples of network performance on cluttered objects: Robots are detected well, even if cluttered or fallen over. The images above are from the validation set and a single anchor box is used for all classes

network, where the feature map has the smallest spatial dimensions, has the largest number of filters. This has obvious advantages when it comes to memory consumption and computational efficiency. In our experiments, we decided to push this feature even further, using few and shallow layers to quickly downscale the feature map, but using a larger number of deep convolutional layers at the lowest level, followed by a similarly shallow and quick upscaling. In Sect. 6 we demonstrate that this network structure is much more efficient, and provides better accuracy for lower computational cost. Figure 1c illustrates our new alternative, the "Pot-Bellied" (ROBO-UNet) architecture.

We also created a second variant of ROBO-UNet (Fig. 1e), where the encoder part of the network consists of successive downscaling layers, as shown in ROBO. To compensate, we increased the number of layers in the pot-belly, although we halved the feature count. We also enhance the decoder part of the network: In the v2 model, the skip connection concatenates feature maps instead of adding them, effectively doubling the feature count. Also, the final classifier is a 3×3 convolution, which increases the network's field of view.

4 Training

We now present our approach to a two-phase training procedure for the network and discuss the virtues of our datasets. We also justify our choice of hyperparameters. Our first chase for training is pre-training on a large synthetic dataset, later we fine-tune the network on a smaller dataset consisting of real images. We also employ a technique called synthetic transfer learning to reduce the number of hand-labeled training examples needed to train the models without overfitting.

4.1 Dataset

The synthetic dataset was created using the Unreal Engine project published by Hess et. al. [16]. First, we created 5000 test images, using 500 scene parameter (carpet color, lighting, color temperature, etc.) variations, and ten different scene arrangements for each. Then, we created a test set with 1250 images using 250 scene parameter sets and five arrangements per set. Both the parameter sets and the arrangements were generated randomly. Annotations were generated for each image automatically using the object label map generated by the engine. Bounding boxes below a size threshold of seven pixels were discarded. The images are 512×384 resolution and are converted to the YUV color space.

Synthetic images are an excellent way of pre-training a network on a large dataset, yet due to the differences between a synthetic and a real environment we require a database of real images to fine-tune the network. But the pre-training allows a much smaller dataset for fine-tuning than would be otherwise required. For these reasons, we created a real semantic segmentation database consisting of 900 images taken at five separate locations: at the venues of RoboCup17/18/19, at the venue of IJCAI17 and in our laboratory at Griffith University. We also have a database for detection containing a grand total of 2100 images from the same five locations.

We manually annotated the images using a tool of our own creation [27]. Our tool provides several ways to aid the annotation process, such as tools for drawing polygons and lines, as well as square and circular brush tools. The program uses the superpixel segmentation method proposed by Li and Chen [45] to speed up the labeling process. In the case of successive images, the tool is able to use dense optical flow to approximate the labels of the next image. Using the tool, it is also possible to mark the edges of the field, setting pixels and labels outside the field to black and background, respectively. This dataset can be used for detection easily by computing bounding boxes for the connected label components.

Despite having a fair number of real images, they were considerably less varied than the synthetic images, since they included only three locations with their unique environmental settings (such as lighting and carpet color). To compensate for this disadvantage, we used data augmentation techniques, such as flipping images horizontally. To emulate changes in lightning conditions, we applied random changes in the brightness, contrast, hue and saturation of the images. To introduce further variation into the dataset, we also applied random affine transformations on the images and the labels.

4.2 Training procedure

The popular PyTorch framework was used for training the network, using the built-in ADAM optimizer for all

Table 1 Hyperparameters used for the training procedures

Param	LR	η_{min}	Decay	β	n	N
Segmentation	1e−3	1e−4	1e−6	0.9	64	100
S. Fine-tune	1e−4	1e−5	1e−5	0.9	16	200
Detection	1e−3	5e−5	1e−5	0.9	64	125
D. Fine-tune	1e−4	5e−6	1e−4	0.9	64	125

N and n denote the number of epochs and the batch size, β is the momentum parameter

models. During training, we used a cosine annealing-based learning rate schedule, decreasing the rate to $\eta_{\min}$. Table 1 displays the hyperparameters used for pre-training and fine-tuning the semantic segmentation and label propagation networks.

The most important change of our training method is the application of L1 regularization to the weights [46]. While L1 regularization is primarily a way to avoid overfitting, it has a desirable side effect: sparsifying the network's weight matrices. While most state-of-the-art methods of pruning devote considerable computational expense to find the least influential weights, using L1 regularization ensures that the majority of the weights are already effectively zero and can be pruned without affecting the network at all [46].

Still, these weights do not become exactly zero, therefore deleting them may still cause a minor disturbance. Therefore, after setting them to zero, we fine-tune the network for another 25 epochs using an $\eta_{\min}/2$ learning rate, while forcing the pruned weights to remain zero. The weights to be pruned are selected independently for each layer by comparing the magnitude of the weight to the largest absolute weight in the same layer. For our results, we set the threshold to 0.01.

By controlling the relative weight of the regularization term, we can influence the ratio of pruned weights. By changing this weight, we trained six different versions of the same network. In the first version, we do use regularization, while in the other five, we steadily increased the regularization weight, eventually achieving as much as 97% on the ROBO-BN model. Note that the weights to be pruned are not selected evenly from all layers, therefore pruning $x\%$ of the weights does not result in an execution-time reduction of $x\%$.

4.3 Synthetic transfer

One of the major challenges of supervised learning is the need to use large training databases to avoid overfitting. It is well known, however, that these problems can be mitigated via transfer learning, where the neural network is first pre-trained on a large database for a different, but similar task [47]. This allows the network to develop a basic understanding of images, which it can apply to other image-based tasks. Then, the last few layers of the network are fine-tuned on a much smaller database for the desired task. This scheme works mainly because the first part of the network performs generic feature extraction, while the last parts of the network are more task specific. By retraining the last part only, we can train the network for a different task (as long as the low level features are useful for this task as well), while the amount of free parameters is considerably less, allowing the use of a much smaller database of hand-labeled images.

Our training scheme is somewhat similar to transfer learning, in that we first pre-train the network on a large database, then we fine-tune it on a smaller one. Make the observation that, in our case, both databases are for the same object detection task, but emerge from different sources. As high quality and realistic the synthetic images may be, the distribution of their pixel values is fundamentally different from the real images. Also, the real images have complex, cluttered backgrounds, which can easily be confused with relevant foreground classes.

For the above reasons, we propose a radically different transfer learning scheme, in which the first few layer weights are retrained on the second database. This is in sharp contrast to vanilla transfer learning of the last few layers. We argue that this scheme is reasonable, since the first few layers of the network are responsible for extracting features from the image. We might also want to fine-tune middle-level layers to allow the network to learn more complex backgrounds. Note that in most convolutional neural networks (including ROBO), the first few layers of the network contain much fewer parameters than the last few. This allows us to retrain more layers with similar amounts of data without overfitting, than in vanilla transfer learning.

5 Implementation

To ensure real-time performance of our object detection pipeline, we must employ several techniques to improve the speed of the trained neural network. For the Nao hardware, these improvements are critical. The networks resulting following the training as described in the previous section require approximately 1 second per image to run on a Nao V5 robots using the Darknet library [18]. This performance remains an unacceptable frame rate.

Moreover, our vision pipeline includes a handcrafted field detection system, which is used by our network to crop part of the image. (The outside the field is usually the top part of the image.) This approach comes with two advantages. First, it reduces the number of pixels to be processed without reducing the level of detail. Second, if the network is trained on images where the parts outside the field are omitted, it avoids learning complex backgrounds outside the field (which are easily confused with field objects). While this technique provides considerable improvement in the network's speed and suitable for the participation in the competition, this improvement is highly dependent on the robot's position in the field. For this reason, and for fair judgement of the merit of the tailor-made design, in what follows we used uncropped images

when comparing the execution times of different models and methods.

5.1 RoboDNN

While implementation could use an existing library, the target platform (Nao robot) is a challenge. For example, Caffe is a relatively old library with numerous dependencies, making it difficult to compile for the Nao robot. While the newer Darknet [18] has no dependencies, it lacks support for several important features we used in our design, such as dilated convolutions and affine batch normalization. Thus, we created our own C++ library called RoboDNN, based on Darknet, implementing the most common neural network layers. Our library is designed for inference only, therefore all code for training the networks was stripped. Our library has no external dependencies, does not require C++11, and - like all of MiPal's code - compiles using the strictest compiler settings.

The current version of RoboDNN is compatible with PyTorch. Our code includes support for dilated convolutions, output padding for transposed convolutions, and layers for affine batch normalization. Thus, RoboDNN is fully compatible with neural networks trained in PyTorch, and we provide the code to export the weights of PyTorch models along with the library. Our library is also optimized for maximum efficiency, including support for accelerating pruned networks, running on cropped images and several in-place operations for memory efficiency.

5.2 Label propagation

Our other technique for increasing the speed of our pipeline is label propagation. Here, we estimate the labels of the next image by using the labels of the previous one. We can achieve a considerable increase in speed by only running the main neural network every ten frames (the Nao robot's camera supports 30 FPS) and provided that accurate label propagation can be implemented using a significantly faster algorithm.

We employed Gunnar Farneback's dense optical flow algorithm [48] to move the labels to their new location. While this solution is able to improve the vision pipeline's average frame-rate considerably, it has a few shortcomings. Namely, small, single-pixel errors would accumulate over time, constantly eroding small objects, making the re-run of the neural network necessary after a certain number of frames. Moreover, optical flow is known to struggle with faster movements, nor can it detect new objects appearing in the image (or partially seen objects sliding in).

6 Experimental results

We evaluated the trained networks on both datasets, computing the mean average precision (mAP) of the detections. For the semantic segmentation networks, we ran connected component labeling on the predicted segmentation to get detections. This is done because our aim is to derive a common measure for both segmentation and detection networks. Moreover, we argue that achieving a good mAP score is more important in the robot soccer setting than extracting accurate shape information. For a comprehensive comparison, Table 4 also provides the Intersection over Union (IoU) metrics for the segmentation networks.

We compared the Tiny YOLO v3 and the standard U-Net networks against our proposed architectures. We also examined the effect of pruning. We used three model versions: ROBO, ROBO-BN and ROBO-2C. ROBO-2C is identical to ROBO, except the first layer only has two input channels $[Y, (U+V)/2]$. We determined the approximate number of operations required to run these models and measured the average achievable frame-per-second (FPS) value of the entire vision pipeline on the Nao v6 robot using a single core.

6.1 Comparison of models

We evaluated the models using different IoU threshold values. Each threshold value corresponds to the minimum IoU value between the predicted and the ground truth bounding boxes required to consider that a proposed rectangle is a correct detection. Lower threshold values mean that the evaluation is more lenient toward inaccurate localization. However, there is a slight problem with this method. In the case of tiny objects (such as the ball, and even more so the line crossing), even small errors in the localization can drastically decrease the IoU value. This potential bias will cause the evaluation method to disproportionately punish localization errors as opposed to classification or confidence errors.

To remedy this, we also compute the mAP values using a different error measure for localization, namely the Euclidean distance between the bounding box centers. It is worth mentioning that this criterion ignores errors of the bounding box shape, although this is a relatively minor issue considering the rigidity of the detected objects. For semantic segmentation, we use the IoU between segments and the distances between segment centers. Tables 2 and 3 show the results on the synthetic and the real datasets, respectively.

The results show that ROBO-UNet outperforms UNet on both datasets, and ROBO achieves higher detection

Table 2 mAP comparison on the synthetic database using the IoU and the pixel distance metrics

IoU	0.75	0.5	0.25	0.1	0.05
Tiny YOLO	**29**	**65**	71	70	70
ROBO	11	44	67	74	77
ROBO-BN	17	53	**73**	**81**	**82**
ROBO-2C	11	44	65	75	77
UNet	57	70	78	84	85
ROBO-UNet	**61**	**73**	**81**	**87**	**88**
ROBO-UNet-v2	**61**	72	80	86	86
Distance (px)	4	8	16	32	64
Tiny YOLO v3	**60**	70	70	70	70
ROBO	33	66	85	89	89
ROBO-BN	45	**76**	**87**	**90**	**90**
ROBO-2C	33	66	84	88	88
UNet	77	83	86	87	88
ROBO-UNet	**81**	**86**	**89**	**90**	**90**
ROBO-UNet-v2	80	85	88	89	89

Best values for detection and segmentation are shown in bold

Table 3 mAP comparison on the real database using the IoU and the pixel distance metrics

IoU	0.75	0.5	0.25	0.1	0.05
Tiny YOLO	**21**	**65**	69	65	64
ROBO	6	45	72	79	80
ROBO-BN	11	55	**80**	**85**	**86**
ROBO-2C	8	49	74	80	81
UNet	62	76	82	84	84
ROBO-UNet	**66**	**81**	**86**	**88**	**88**
ROBO-UNet-v2	59	76	83	84	85
Distance (px)	4	8	16	32	64
Tiny YOLO v3	**41**	**64**	69	65	64
ROBO	14	46	77	84	84
ROBO-BN	22	57	**82**	**87**	**87**
ROBO-2C	16	50	77	84	85
UNet	68	76	82	84	86
ROBO-UNet	**73**	**81**	**85**	**89**	**90**
ROBO-UNet-v2	67	76	83	85	87

Best values for detection and segmentation are shown in bold

scores than Tiny YOLO when the localization accuracy requirement is loose. Importantly, ROBO outperforms Tiny YOLO more decisively on the real dataset. This is most likely due to the fact that Tiny YOLO has several times more parameters, making it much easier to overfit on a small database. Also, ROBO-UNet outperforms both UNet and ROBO-UNet-v2 quite decisively.

In all four cases the ROBO detection architectures respond to the change in the strictness of the localization criterion much more drastically, suggesting that they struggle more with accurate localization, while Tiny YOLO struggles with confidence and classification. This is underscored by the fact that using a localization criterion that is loose enough, ROBO-based models invariably manage to outperform Tiny YOLO. We believe that this is largely due to the problem-specific output generation methods employed by ROBO, since this phenomenon does not appear between UNet and ROBO-UNet.

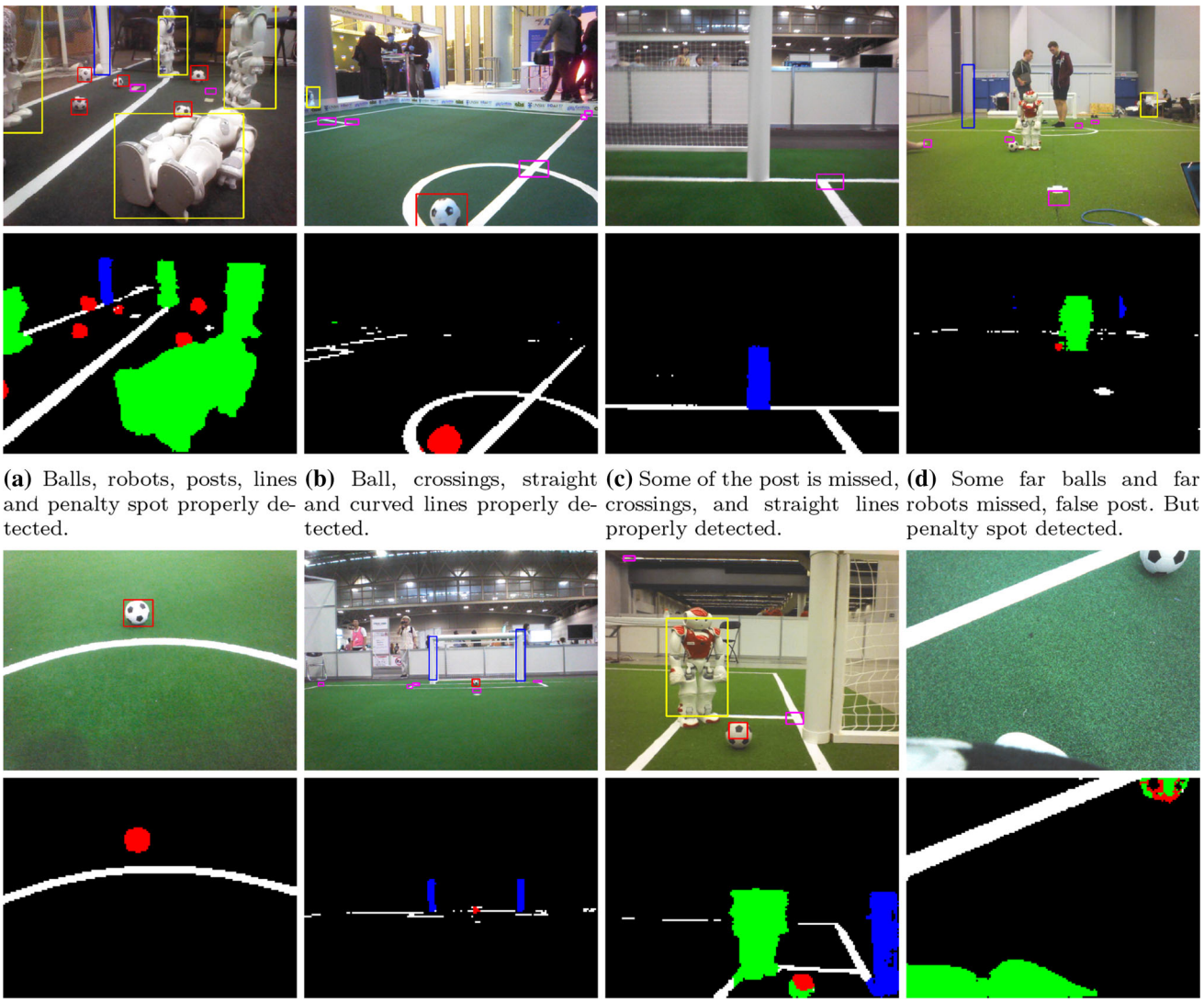
The breakdown of ROBO variants is quite predictable: The ROBO-2C variant performs similarly to ROBO, while the somewhat more complex ROBO-BN outperforms the other two detection methods, although ROBO-2C manages to achieve similar performance on the synthetic dataset with loose criteria. This suggests that higher parameter numbers help with accurate localization. Also, the two-channel version falls short of the other two by a few percentage points, but still manages to clearly outperform Tiny YOLO. Figure 3 shows some example results of ROBO on the synthetic test dataset, and also some good and bad results on the real test dataset.

The per-class mAP results show that the network's performance does not depend strongly on the size of the objects. The ROBO model achieves the highest mAP on the ball and goalpost classes (89 on both), while it struggles more with the crossing and robot classes (80 and 76). Interestingly, the largest class appears to have the smallest average precision. As demonstrated in Fig. 3, the networks are able to detect objects at a fair distance, although ROBO's localization is somewhat inaccurate with small objects. Recall, that qualitative detection is more important than accurate localization, especially for small objects, since they are far.

6.2 Effects of pruning

Figure 4 shows the effect of pruning on accuracy, as well as the number of operations required for each model version. Our results show that it is possible to prune approximately 90% of the ROBO model's parameters with only a negligible drop in accuracy, while reducing the number of operations by 80%. The ROBO-2C architecture retains its accuracy better than ROBO, making this model a superior choice. Also, the share of parameters pruned from ROBO-BN is higher, since it has approximately twice the number of parameters. The drop in performance is also more steep in this case, making this model slightly inferior. Notably, ROBO-UNet-v2 is considerably less affected by pruning, yielding a model that outperforms ROBO-UNet slightly with 10% less operations.

We also considered an alternate version of the ROBO detection networks that runs on half-resolution (256×192) input images. These architectures are marked with the **HR**



(a) Balls, robots, posts, lines and penalty spot properly detected.

(b) Ball, crossings, straight and curved lines properly detected.

(c) Some of the post is missed, crossings, and straight lines properly detected.

(d) Some far balls and far robots missed, false post. But penalty spot detected.

(e) Nearby ball and curved line properly detected. No false positives.

(f) Far balls, posts, crossings and lines properly detected. No false positives.

(g) Ball misclassified as robot. Robot, crossings, post and lines properly detected.

(h) Ball misclassified as robot. Robot feet and line properly detected.

Fig. 3 Some good (left) and bad (right) detection results on the real database both in- and outdoors

abbreviation, and lose another $[0.2 - 1.5\%]$ accuracy compared to their full-scale counterparts. The HR models are different from the originals in that the first strided convolution layer is removed from the architecture, leaving the rest unchanged. While this only results in a minor decrease in the number of operations ($10M$), the reduction of run time is more significant due to the inefficiency of convolution on feature maps with large spatial dimensions. Among these models, ROBO-2C-HR achieves the best combination of accuracy and efficiency once again, losing very little in terms of accuracy in exchange for a significant performance boost.

6.3 Effects of synthetic transfer

We also ran experiments with synthetic transfer learning, as shown in Fig. 5. We ran several tests, where we changed the number of initial layers to retrain. By increasing this number, we allow the network to learn the real dataset better, resulting in faster convergence, but the network becomes more likely to overfit. In these experiments, we fine-tuned the rest of the layers using a smaller learning rate (by a factor of ten) instead of freezing them completely.

The results show that retraining the first few layers only can achieve superior results to retraining the entire network, despite using only 0.1–19.9 percent of the

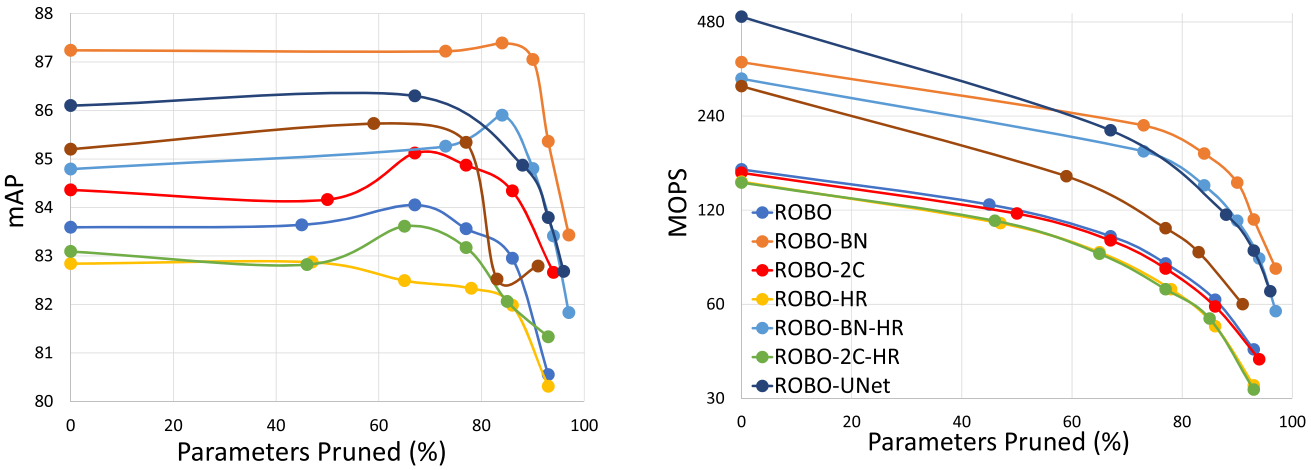


Fig. 4 Effect of pruning on the mAP (left) and the number of operations (right)

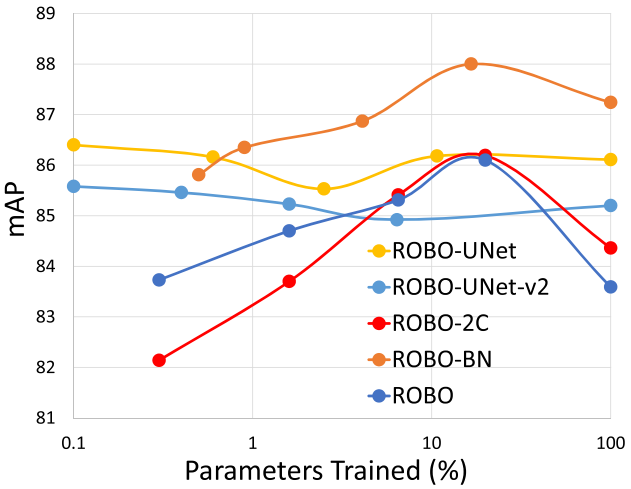


Fig. 5 Effect of synthetic transfer learning on the mAP

parameters respectively. Notably, synthetic transfer learning works better for the ROBO detection architectures, where the mAP peaked [0.8–2.5]% higher than with normal training. For the ROBO-UNet variants, retraining the first one or two layers of the network achieves only marginally ([0.3–0.4]%) better results (Tables 4 and 5).

Table 4 IoU values of the segmentation networks

Model	UNet	ROBO-UNet	ROBO-UNet-v2
Synthetic	72.89	**75.51**	74.7
Real	79.57	83.07	**83.28**
Real (Pruned)	79.97	**81.21**	80.76

Best values are shown in bold

Table 5 Parameter count of different architectures

T.YOLO	ROBO	R-BN	UNet	R-UNet	R-UNetv2
8322 K	557 K	1249 K	99 K	691 K	390 K

The HR and 2C versions have similar number of parameters to their standard counterpart

6.4 Evaluation of run time

We tested the execution time of our entire vision pipeline on a Nao v6 robot, using the top camera image. We used a single core to run the neural network, and we ran the pipeline with other soccer subsystems active. Table 6 shows the execution time of the pruned versions of the models compared in the previous subsection. The results show a clear improvement as a result of pruning, and that both variants of ROBO-UNet outperform the vanilla UNet

Table 6 Run times (t) and accuracies of different architectures

Model	mAP	MOPS	t (ms)	FPS
ROBO-UNet	82.68	66	190	5.3
ROBO-UNet-v2	**82.79**	**60**	**172**	**5.8**
UNet	81.58	88	253	3.95
ROBO	80.55	43	146	6.9
ROBO-HR	80.31	33	77	**13**
ROBO-BN	**83.43**	78	267	3.7
ROBO-BN-HR	81.83	57	133	7.5
ROBO-2C	82.66	40	136	7.4
ROBO-2C-HR	81.33	**32**	**75**	**13.3**
Tiny YOLOv3	65.12	5800	13500	0.1

Best values for detection and segmentation are shown in bold

Label propagation was not used for these measurements

in speed as well. Also, the v2 variant achieves marginally better run time and accuracy after pruning.

Moreover, our detection architectures offer a run time that is a mere fraction of the Tiny YOLO v3. Here, the novel ROBO-2C and ROBO-2C-HR variants offer the two best trade-offs between run time and accuracy. Notably, the HR models run approximately twice as fast, despite needing more than half the number of operations compared to their full resolution counterparts. We believe that this is due to the `im2col` operation, which is a part of the convolution implementation, responsible for arranging the feature map activations in a matrix form. In inputs with large spatial size, this operation requires approximately 50% of the execution time.

The single-core run time of ROBO-2C on the Nao v6 robot is 430 ms, which translates to approximately 2.3 frames per second. With94% pruning, however, the run time decreased to 136 ms, or 7.4 fps, while we see a similar decrease with ROBO-UNet, which takes 740 ms or 1.4 fps with all weights, but after 95% pruning it is reduced to 190 ms or 5.3 fps. Our best performing model is the ROBO-2C-HR, which achieves a reasonable 81.33% mAP, while besting all other models with a run time of 75 ms or 13.3 fps.

6.5 Evaluation of label propagation

In Table 7, we also present the results of label propagation using optical flow on a special database that contains image sequences. Since the Farneback optical flow takes approximately 24 ms to run on the Nao v6 robot, by running the full network every ten frames, we can achieve a run time of 40 ms/25 fps for segmentation, and 29 ms/34.4 fps for detection.

The difference in performance of the models is much more pronounced on the label propagation database. This is partly caused by the small size of this dataset (approx. 275 images), and that most of the images in this set were shot in a single location. Due to these factors, a particular set of scene parameters and setup is over-represented in this dataset, causing some architectures to perform surprisingly well, while others struggle. Invariably, however, using label propagation reduces the mAP by [4–7]%, which we consider acceptable for situations where such high frame rates are needed.

Table 7 Effect of label propagation (LP) on mAP and run time (t)

Model	mAP	mAP (LP)	*t* (ms)	FPS
ROBO-UNet	**78.75**	**74.14**	40	25
ROBO-UNet-v2	77.11	72.53	**39**	**25.5**
ROBO	81.38	75.41	37	27
ROBO-HR	80.35	73.51	**30**	33.3
ROBO-BN	77.17	71.17	49	20.4
ROBO-BN-HR	74.59	68.43	35	28.6
ROBO-2C	**84.92**	**77.54**	36	27.8
ROBO-2C-HR	74.97	69.23	**30**	**33.9**

Best values for detection and segmentation are shown in bold

7 Conclusion

In this paper, we presented a fully deep neural network-based object detection framework capable of detecting simultaneously all relevant objects in robot soccer environments. We proposed two new architectures, ROBO and ROBO-UNet, and showed that they outperform Tiny YOLO and UNet both in terms of speed and accuracy. We showed that these models can be trained using synthetic transfer learning to reduce the amount of data required. Our work also produced a small database of real images and a light-weight neural network inference library other researchers can use freely. Our code, the pre-trained models, training and validation datasets and the RoboDNN library are published online [26].

Yet, there are a few possibilities to improve our models. One of them is improving the regularization method, by using group L1 regularization [46]. This would allow us to prune entire convolutional filters instead of individual weights, which in turn would make the execution of the neural network considerably more efficient. In the future, we plan to explore the use of more advanced neural network architectures, such as CapsNets [44], which encode 3D geometry in the network architecture, allowing the network to learn features invariant to viewpoint transformations. This could allow us to use even smaller networks, reducing run time further, while improving generalization.

It is worth noting that while our research was primarily focused on robot soccer, the proposed ROBO architecture can also be applied in other semi-controlled environments, where restrictions on the number, proximity, or other properties of the objects are known beforehand. Industrial settings, such as pipeline monitoring or optometric measurement of animals, are excellent examples for such tasks.

Acknowledgement The research was supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program, and the National Research, Development and Innovation Fund (TUDFO/51757/2019-ITM, Thematic Excellence Program).

Funding Open access funding provided by Budapest University of Technology and Economics.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Mackworth AK (1993) On seeing robots. Technical report, Vancouver, BC, Canada
2. Pulli K, Baksheev A, Kornyakov K, Eruhimov V (2012) Real-time computer vision with openCV. Commun ACM 55(6):61. https://doi.org/10.1145/2184319.2184337
3. Schwarz I, Hofmann M, Urbann O, Tasse S (2015) RoboCup 2015: robot world cup XIX. Springer, New York, pp 239–250
4. Metzler S, Nieuwenhuisen M, Behnke S (2012) RoboCup 2011: robot soccer world cup XV, pp 149–161
5. Albani D, Youssef A, Suriani V, Nardi D, Bloisi DD (2017) In: Behnke S, Sheh R, Sarıel S, Lee DD (eds) RoboCup 2016: robot world cup XX, pp 392–403. Springer, Cham
6. Javadi M, Azar SM, Azami S, Ghidary SS, Sadeghnejad S, Baltes J (2018) In: Akiyama H, Obst O, Sammut, Tonidandel F (eds) RoboCup international symposium, pp 338-349. Springer, Cham
7. Cruz N, Lobos-Tsunekawa K, del Solar JR (2017) In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds) RoboCup international symposium, pp 19–30. Springer, Cham
8. O'Keeffe S, Villing R (2017) In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds) RoboCup international symposium, pp 398–409. Springer, Cham
9. (2017) In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds) RoboCup international symposium, pp 45–58. Springer, Cham
10. Gabel A, Heuer T, Schiering I, Gerndt R (2019) In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: robot world cup XXII, pp 181–192. Springer, Cham
11. Felbinger GC, Göttsch P, Loth P, Peters L, Wege F (2019) In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: robot world cup XXII, pp 150–161. Springer, Cham
12. Speck D, Barros P, Weber C, Wermter S (2017) In: Behnke S, Sheh R, Sarıel S, Lee DD (eds) RoboCup 2016: robot world cup XX, vol 55, pp 19–30. Springer, Cham. https://doi.org/10.1145/2184319.2184337
13. Speck D, Bestmann M, Barros P (2019) In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: robot world cup XXII, vol 55, pp 337–348. Springer, Cham. https://doi.org/10.1145/2184319.2184337
14. Houliston T, Chalup SK (2019) Visual mesh: real-time object detection using constant sample density. In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: robot world cup XXII. RoboCup 2018. Lecture notes in computer science, vol 11374. Springer, Cham. https://doi.org/10.1007/978-3-030-27544-0_4
15. Leiva F, Cruz N, Bugueño I, Ruiz-del Solar J (2019) In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: robot world cup XXII, pp 122-134. Springer, Cham
16. Hess T, Mund M, Weis T, Ramesh V (2017) In: Akiyama H, Obst O, Sammut C, Tonidandel F (eds) RoboCup international symposium, pp 33–44. Springer, Cham
17. van Dijk SG, Scheunemann MM (2019) Deep learning for semantic segmentation on minimal hardware. In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: robot world cup XXII. RoboCup 2018. Lecture notes in computer science, vol 11374. Springer, Cham. https://doi.org/10.1007/978-3-030-27544-0_29
18. Redmon J, Divvala S, Girshick R, Farhadi A (2016) In: Proceedings of the 2016 IEEE conference on computer vision and pattern recognition (CVPR)(IEEE), pp 779–788. https://doi.org/10.1109/cvpr.2016.91
19. Ronneberger O, Fischer P, Brox T (2015) In: International conference on medical image computing and computer-assisted intervention, pp 234–241
20. Reyes E, Gómez C, Norambuena E, Ruiz-del Solar J (2019) In: Holz D, Genter K, Saad M, von Stryk O (eds) RoboCup 2018: robot world cup XXII, vol 55, pp 287–298. Springer, Cham. https://doi.org/10.1145/2184319.2184337
21. Thommen GK, Bouffanais R (2019) Self-organizing maps for storage and transfer of knowledge in reinforcement learning. Adapt Behav 27(2):111. https://doi.org/10.1177/1059712318818568
22. Torrey L, Shavlik J, Walker T, Maclin R (2006) In: Fürnkranz J, Scheffer T, Spiliopoulou M (eds) Machine learning: ECML 2006, vol 27, pp 425–436. Springer, Berlin. https://doi.org/10.1177/1059712318818568
23. Verbancsics P, Stanley KO (2010) In: Fürnkranz J, Scheffer T, Spiliopoulou M (eds) Proceedings of the 12th annual conference on genetic and evolutionary computation, GECCO '10, vol 27, pp 547–554. ACM, New York. https://doi.org/10.1145/1830483.1830587
24. Inoue T, Choudhury S, De Magistris G, Dasgupta S (2018) In: Proceedings of the 2018 25th IEEE international conference on image processing (ICIP), pp 2725–2729. https://doi.org/10.1109/ICIP.2018.8451064
25. Redmon J, Farhadi A (2018) YOLOv3: an incremental improvement. https://arxiv.org/abs/1804.02767
26. Szemenyei M. Robo. https://github.com/szemenyeim/ROBO
27. Szemenyei M, Estivill-Castro V (2019) In: RoboCup 2019: robot world cup XXIII. https://2019.robocup.org/downloads/program/SzemenyeiEstivillCastro2019.pdf
28. Cordts M, Omran M, Ramos S, Rehfeld T, Enzweiler M, Benenson R, Franke U, Roth S, Schiele B (2016) In: Proceedings of the 2016 IEEE conference on computer vision and pattern recognition (CVPR) (IEEE), pp 3213–3223. https://doi.org/10.1109/cvpr.2016.350
29. Jyotiyana M, Kesswani N (2020) Deep learning and the future of biomedical image analysis, pp 329–345. Springer, Cham. https://doi.org/10.1007/978-3-030-33966-1_15
30. Ren S, He K, Girshick R, Sun J (2017) Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 39(6):1137. https://doi.org/10.1109/tpami.2016.2577031
31. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC (2016) SSD: single shot multibox detector. In: Leibe B, Matas J, Sebe N, Welling M (eds) Computer vision – ECCV 2016. ECCV2016. Lecture notes in computer science, vol 9905. Springer, Cham. https://doi.org/10.1007/978-3-319-46448-0_2
32. Redmon J, Farhadi A (2016) YOLO9000: better, faster, stronger. https://arxiv.org/abs/1612.08242v1

33. Long J, Shelhamer E, Darrell T (2015) In: Proceedings of the 2015 IEEE conference on computer vision and pattern recognition (CVPR) (IEEE), pp 3431–3440. https://doi.org/10.1109/cvpr.2015.7298965
34. Chen LC, Papandreou G, Schroff F, Adam H (2018) DeepLab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. In: IEEE transactions on pattern analysis and machine intelligence, vol 40, no 4, pp 834–848. https://doi.org/10.1109/TPAMI.2017.2699184
35. Chen L, Papandreou G, Schroff F et al (2017) Rethinking atrous convolution for semantic image segmentation [J]. Comput Vision Pattern Recog. https://arxiv.org/abs/1706.05587
36. Zheng S, Jayasumana S, Romera-Paredes B, Vineet V, Su Z, Du D, Huang C, Torr PHS (2015) In: Proceedings of the 2015 IEEE international conference on computer vision (ICCV) (IEEE), pp 1529–1537. https://doi.org/10.1109/iccv.2015.179
37. He K, Gkioxari G, Dollar P, Girshick R (2017) In: Proceedings of the 2017 IEEE international conference on computer vision (ICCV) (IEEE), pp 2980–2988. https://doi.org/10.1109/iccv.2017.322
38. Yosinski J, Clune J, Bengio Y, Lipson H (2014) In: Proceedings of the 27th international conference on neural information processing systems, NIPS'14, Vol 2, pp 3320–3328. MIT Press, Cambridge. http://dl.acm.org/citation.cfm?id=2969033.2969197
39. Weiss K, Khoshgoftaar TM, Wang D (2016) A survey of transfer learning. J Big Data 3(1):9. https://doi.org/10.1186/s40537-016-0043-6
40. Sun R, Zhu X, Wu C, Huang C, Shi J, Ma L (2019) The IEEE conference on computer vision and pattern recognition (CVPR), pp 4360–4369
41. Gaier A, Ha D (2019) Weight agnostic neural networks. https://arxiv.org/abs/1906.04358
42. Inoue T, Choudhury S, Magistris GD, Dasgupta S (2018) In: Proceedings of the 2018 25th IEEE international conference on image processing (ICIP) (IEEE), pp 2725–2729. https://doi.org/10.1109/icip.2018.8451064
43. Lin TY, Maire M, Belongie S, Bourdev L, Girshick R, Hays J, Perona P, Ramanan D, Zitnick CL, Dollár P (2015) Microsoft coco: common objects in context
44. Sabour S, Frosst N, Hinton GE (2017) Dynamic routing between capsules. in: Proceedings of the 31st international conference on neural information processing systems (NIPS), pp 3859–3869
45. Li Z, Chen J (2015) IEEE conference on computer vision and pattern recognition (CVPR) 1356–1363
46. Scardapane S, Comminiello D, Hussain A, Uncini A (2017) Neurocomputing 241:81. https://doi.org/10.1016/j.neucom.2017.02.029
47. Shin H, Roth HR, Gao M, Lu L, Xu Z, Nogues I, Yao J, Mollura D, Summers RM (2016) Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE Trans Med Imaging 35(5):1285. https://doi.org/10.1109/TMI.2016.2528162
48. Farnebäck G (2003) In: Bigun J, Gustavsson T (eds) Scandinavian conference on image analysis, pp 363–370. Springer, Berlin Heidelberg