





Deep Learning for Semantic Segmentation on Minimal Hardware

Sander G. van Dijk^(✉)  and Marcus M. Scheunemann 

University of Hertfordshire, Hertfordshire AL10 9AB, UK
sgvandijk@gmail.com

Abstract. Deep learning has revolutionised many fields, but it is still challenging to transfer its success to small mobile robots with minimal hardware. Specifically, some work has been done to this effect in the RoboCup humanoid football domain, but results that are performant and efficient and still generally applicable outside of this domain are lacking. We propose an approach conceptually different from those taken previously. It is based on semantic segmentation and does achieve these desired properties. In detail, it is being able to process full VGA images in real-time on a low-power mobile processor. It can further handle multiple image dimensions without retraining, it does not require specific domain knowledge to achieve a high frame rate and it is applicable on a minimal mobile hardware.

Keywords: Deep learning · Semantic segmentation · Mobile robotics · Computer vision · Minimal hardware

1 Introduction

Deep learning (DL) has greatly accelerated progress in many areas of artificial intelligence (AI) and machine learning. Several breakthrough ideas and methods, combined with the availability of large amounts of data and computation power, have lifted classical artificial neural networks (ANNs) to new heights in natural language processing, time series modelling and advanced computer vision problems [11]. For computer vision in particular, networks using convolution operations, i.e., *Convolutional Neural Networks* (CNNs), have had great success.

Many of these successful applications of DL rely on cutting edge computation hardware, specifically high-end GPU processors, sometimes in clusters of dozens to hundreds of machines [15]. Low-power robots, such as the robotic footballers participating in RoboCup, are not able to carry such hardware. It is not a surprise that the uptake of DL in the domain of humanoid robotic football has lagged behind. Some demonstrations of its use became available recently [1, 5, 8, 14, 16]. However, as we will discuss in the next section, these applications are so far rather limited; either in terms of performance or in terms of their generalisability for areas other than RoboCup.

In this paper, we will address these issues and present a DL framework that achieves high accuracy, is more generally applicable and still runs at a usable frame rate on minimal hardware.

The necessary conceptual switch and main driver behind these results is to apply DL to the direct semantic segmentation of camera images, in contrast to most previous work in the humanoid robotic football domain that has applied it to the final object detection or recognition problem. Semantic segmentation is the task of assigning a class label to each separate pixel in an image, in contrast to predicting a single output for an image as a whole, or some sub-region of interest. There are three primary reasons why this approach is attractive.

Firstly, semantic segmentation networks can be significantly smaller in terms of learnable weights than other network types. The number of weights in a convolution layer is reduced significantly compared to the fully connected layers of classical ANNs, by ‘reusing’ filter weights as they slide over an image. However, most image classification or object detection networks still need to convert a 2D representation into a single output, for which they do use fully connected layers on top of the efficient convolution layers. The number of weights of fully connected layers is quadratic in their size, which means they can be responsible for a major part of the computational complexity of the network. Semantic segmentation networks on the other hand typically only have convolution layers—they are *Fully Convolutional Networks* (FCNs)—and so do away with fully connected ones, and the number of their weights only grows linearly with the number of layers used.

Secondly, the fully convolutional nature also ensures that the network is independent of image resolution. The input resolution of a network with a fully connected output layer is fixed by the number of weights in that layer. Such a network, trained on data of those dimensions, cannot readily be reused on data of differing sizes; the user will have to crop or rescale the input data, or retrain new fully connected layers of the appropriate size. Convolution operations on the other hand are agnostic of input dimensions, so a fully convolutional network can be used at any input resolution¹. This provides very useful opportunities. For example, if a known object is tracked, or an object is known to be close to the camera, the algorithm allows for an on-demand up and down scaling of vision effort. Instead of processing a complete camera frame when searching for such an aforementioned object, only an image subset or a downsampled version of a camera frame is processed.

Finally, semantic segmentation fits in directly with many popular vision pipelines used currently in the RoboCup football domain. Historically, the domain consisted of clearly colour coded concepts: green field, orange ball, yellow/blue goalposts. Commonly a lookup-table based approach is used to label each pixel separately, after which fast specialised connected component, scanning, or integral image methods are applied to detect and localise all relevant objects. Over the years the scenario has developed to be more challenging (e.g.,

¹ Given that the input dimensions are not so small that any down-sampling operations, e.g. max pooling, would reduce the resolution to nil.

natural light, limited colours) and unstructured, making the lookup-table methods less feasible. Using a semantic segmentation CNN that can learn to use more complex features would allow the simple replacement of these methods and still reuse all highly optimised algorithms of the existing vision pipeline.

2 Related Work

The RoboCup domain offers a long history in research on efficient, practical computer vision methods; already the very first RoboCup symposium in 1997 dealt with “Real-Time Vision Processing for a Soccer Playing Mobile Robot” [4], and the 2016 Kid-Size champions still heavily relied on optimisations, e.g., regions of interest and downscaled images, to make vision viable [2]. To ensure keeping track of and participating in a dynamic game of football, the robots ideally should process at least 20 to 30 fps. However, they only have very limited energy resources yielding minimal computational power for achieving that.

Recent developments in low-power, mobile computational platforms, as well as in efficient deep learning, have now made it possible to adopt DL in small mobile robots. One of the first works on DL in the RoboCup domain presented a CNN trained to separately predict the x and y coordinates of a ball in a camera image [16]. Although this network performed relatively well, it could only process a few images per second and operated on heavily downscaled images. At the same time other authors were able to create a CNN-based system that could recognise Nao robots within milliseconds [1]. However, this method relied on a region proposal preprocessing method very specific to RoboCup. This work was later generalised to a different RoboCup league [10], but still relies on the specifically colour-coded nature of the robot football pitch. Instead, the approach taken in this paper is to use CNNs as the very first processing step, and only *after* that step apply domain specific algorithms. This same approach was taken in a recent work very much related to ours [14], but for large humanoid robots with powerful hardware that cannot feasibly be used by smaller size mobile robots, such as Kid-Size humanoids or perhaps drones.

In recent years there has been a growing body of work on creating small, but capable networks, for enabling their use on more restricted hardware. One approach is to try to minimise the complexity of convolutions by first applying a ‘bottleneck’ 1×1 convolution layer that reduces the number of features used in the actual $N \times N$ convolution. This idea originated with ResNet [7] to help make training of very deep networks feasible, but at the same time can also reduce run time costs. For networks of the sizes used in this paper however, the computational cost of a bottleneck layer outweighed the benefit of a reduced number of features in the subsequent layer. A different idea is to discretise quantities used in the networks, with the idea that integer operations can be much more efficient in low-end computation devices. The culmination of this idea is in network designs such as XNOR-nets [12] that use very basic and fast bitwise operations during prediction, and DoReFa-nets [17] that further extend this idea to training. We do not study such binary nets here, as at the moment there is

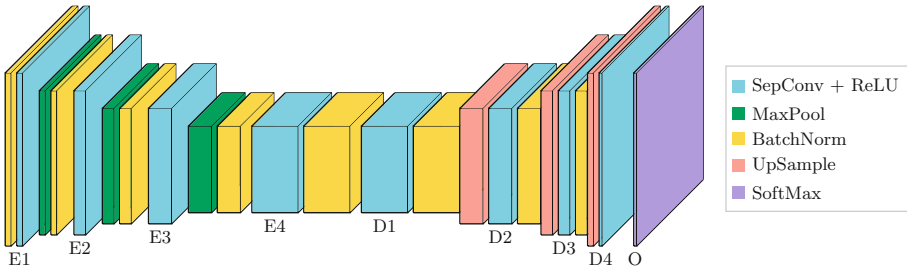


Fig. 1. The architecture of the networks used consists of a series of fully convolutional encoding (E1–E4) and decoding (D1–D4) steps. A pixelwise softmax output layer provides the final classifications. Network variations differ in the actual number of encoding and decoding steps used, filter size and initial depth, filter depth multiplication factor and convolution stride.

no implementation available of the operators required by such networks for the most popular deep learning libraries, and we are interested in systems that can be easily adapted and implemented by the reader using such libraries.

Instead, the optimisations applied to our networks are very much motivated by MobileNets [9]. Most notably, we utilise *depthwise separable convolutions* to significantly reduce the computational complexity of our segmentation networks. Such convolutions split a regular convolution into a filter and a combination step: first a separate 2D filter is applied to each input channel, after which a 1×1 convolution is applied to combine the results of these features. This can be seen as a factorisation of a full convolution that reduces the computational cost by a factor of $\frac{1}{N} + \frac{1}{K^2}$, where N is the number of output features and K the kernel size. Not all convolutions can be factorised like this, so separable convolutions have less expressive power, but the results of the original MobileNets and those reported here show they can still perform at high accuracy.

3 Network Architecture

As mentioned before, our approach is based on fully convolutional semantic segmentation networks. The main structure of our networks is similar to popular encoder-decoder networks, such as U-Net [13] and SegNet [3], mainly following the latter. In such networks, a first series of convolution layers encode the input into successively lower resolution but higher dimensional feature maps, after which a second series of layers decode these maps into a full-resolution pixelwise classification. This architecture is shown in Fig. 1.

SegNet and U-Net both have the property that some information from the encoder layers are fed into the respective decoder layers of the same size, either in terms of maxpooling indices, or full feature maps. This helps overcoming the loss of fine detail caused by the resolution reduction along the way. As good performance is still possible without these connections, we do not use those here. They in fact introduce a significant increase in computation load on our

hardware, due to having to combine tensors in possibly significantly different memory locations.

Another modification is to use depthwise separable convolution, as introduced by MobileNets [9], as a much more efficient alternative to full 3D convolution. This is one of the major contributions to efficiency of our networks, without significantly decreasing their performance.

To study the trade-off between network minimalism, efficiency and performance, we create, train and evaluate a number of varieties of the above network, with different combinations of the following parameter values:

1. **Number of Layers (L)**— $L \in \{3, 4\}$, the number of encoding and decoding layers. We always use the same number of encoding and decoding layers.
2. **Number of Filters (F)**— $F \in \{3, 4, 5\}$, the number of filters used in the first encoding layer.
3. **Filter Multiplier (M)**— $M \in \{1.25, 1.5, 2\}$, the factor by which the number of filters increases for each subsequent encoding layer, and decreases for each subsequent decoding layer.
4. **Convolution Stride (S)**— $S \in \{1, 2\}$, the stride used in each convolution layer.

Larger and smaller values for these parameters have been tried out, but we only report those here that resulted in networks that were able to learn the provided task to some degree, but were light enough to be run on the minimal test hardware. Specific instantiations will be denoted with $L_x F_y M_z S_w$ with parameter values filled into the place holders. For instance, $L_3 F_4 M_{1.25} S_2$ is the network with 3 encoding and decoding layers, 4 features in the first feature map, a multiplier of 1.25 (resulting in 4, 5 and 6 features in each subsequent layer) and a stride of 1. Not all combinations are valid: a combination of $L = 4$ and $S = 2$ would result in invalid feature map sizes given our input of 640×480 images. The total number of network types then is 27. Finally, all convolution layers use 3×3 filters, padding to have output size the same as input size and no bias.

4 Experiments

The networks described in the previous section are trained to segment ball pixels in images taken by a Kid-Size RoboCup robot on a competition pitch. Specifically, the image set `bitbots-set00-04` from the Bit-Bots’ Imagetagger² was used. It contains 1000 images³ with 1003 bounding box ball annotations. To derive the target pixel label masks for training the networks, the rectangular annotations are converted to filled ellipsoids. Figure 2 shows an example of the input images and targets.

We use the TensorFlow library to construct, train and run the networks. The networks are trained on an NVIDIA GeForce GTX 1080-ti GPU, with a

² <https://imagetagger.bit-bots.de/images/imageset/12/>.

³ Images taken at the 2016 world championship in Leipzig, Germany.

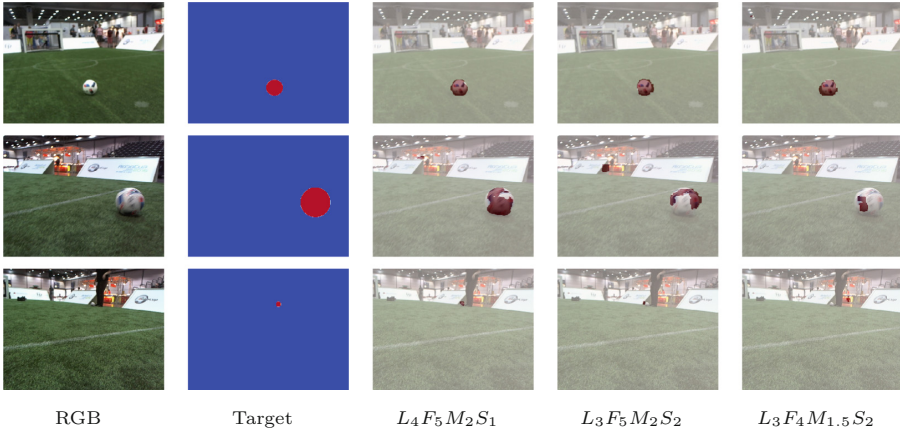


Fig. 2. Examples of input, target and segmentation outputs. The outputs are respectively of the best stride 1 network, the best stride 2 network, and the second best stride 2 network that achieves 20 frames per second on QVGA images.

categorical cross-entropy loss function using stochastic gradient descent with a starting learning rate of 0.1, a decay factor of 0.004 and a momentum of 0.9. The dataset is split in a training set of 750 images, a validation set of 150 and a test set of 100 images. The sets are augmented to double their size by including all horizontally mirrored images. During training, a batch size of 10 images is used. Networks are trained for 25 epochs.

For testing the performance of the networks we map the class probabilities from the softmax output to discrete class labels and use this to calculate the commonly used *Intersection over Union* (IoU) score as $\frac{TP}{TP+FP+FN}$, where TP is the number of true positive ball pixels, FP the number of false positives and FN the number of false negatives. Due to the extreme class imbalance, the networks hardly ever predict the probability of a pixel being part of a ball, $P(B)$, to be above 0.5. This means that if we use the *most probable* class as final output, the IoU score often is 0, even though the networks do learn to assign relatively higher probability at the right pixels. Instead we find the threshold θ^* for $P(B)$ that results in the best IoU score for each trained network.

Finally, since the original goal is to develop networks that can run on minimal hardware, the networks are run and timed on such hardware, belonging to a Kid-Size humanoid football robot, namely an Odroid-XU4. This device is based on a Samsung Exynos 5422 Cortex-A15 with 2 GHz and a Cortex-A7 Octa core CPU, which is the same as used in some 2015 model flagship smartphones. Before running the networks, they are optimised using TensorFlow's *Graph Transform* tool, which is able to merge several operations, such as batch normalisation, into more efficient ones. The test program and TensorFlow library are compiled with all standard optimisation flags for the ARM Neon platform. We time the networks both on full 640×480 images and on 320×256 images.

5 Results

We firstly evaluate the performance of semantic segmentation networks trained for the official RoboCup ball. We compare the performance and runtime of the different network instantiations with each other, as well as to a baseline segmentation method. This method is based on a popular fast *lookup table* (LUT) method, where the table directly maps pixel values to object classes. To create the table, we train a *Support Vector Machine* (SVM) on the same set as the CNNs to classify pixels. More complex and performant methods may be chosen, perhaps specifically for the robot football scenario, however we selected this method to reflect the same workflow of training a single model on simple pixel data, without injecting domain specific knowledge. We did improve performance by using input in HSV colour space and applying grid search to optimise its hyper parameters.

Secondly, we extend the binary case and train the networks for balls and goal posts, and compare the network performance with the binary segmentation.

5.1 Binary Segmentation

We first analyse the segmentation quality of the set of networks and the influence of their parameters on their performance. The best network is $L_4F_5M_2S_1$ with a best IoU of 0.804. As may be expected, this is the one with the most layers, most filters, highest multiplication factor and densest stride. The least performant network is one of the simplest in terms of layers and features: $L_3F_3M_{1.25}S_1$ with a best IoU of 0.085. Perhaps surprisingly the version of that network with stride 2 manages to perform better, with a score of 0.39. Figure 3 shows the distributions of performance over networks with the same values for each parameter. One can see that overall more complex networks score higher, but that the median network with stride 2 performs better than the median with stride 1.

Figure 4 compares the runtime and IoU scores for all networks. The data points are grouped by stride, resulting in clearly distinct clusters: as expected the networks with stride 2 have a significantly lower runtime requirement. The timings run from 121 to 397 ms per full 640×480 resolution frame, which is equivalent to approximately 8 and 2.5 frames per second, respectively.

The best performing network in terms of IoU is also the least efficient one, but the second best is a full 74 ms faster with a drop in IoU of only 0.003. The linear fits show that there is indeed a trend within each cluster of better performance given the runtime, but it is clear that this is not generally the case: networks with similar runtimes can vary greatly in achieved performance.

The SVM-based LUT method, though being significantly faster, scores well below most networks, with an IoU of 0.085. This is because such a pixel-by-pixel method does not consider the surrounding area and thus has no way to discern pixels with similar colours, resulting in many false positives for pixels that have colours that are also found in the ball. In contrast, the CNNs can perform much more informed classification by utilising the receptive field around each pixel.

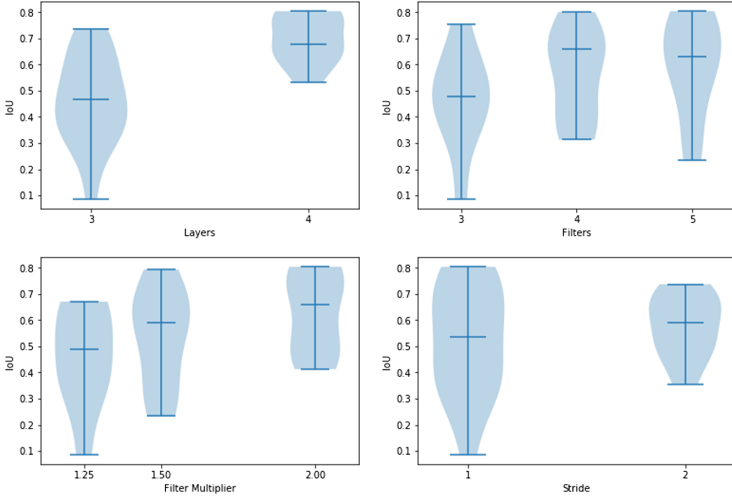


Fig. 3. Performance distributions for each network parameter. Horizontal bars show the minimum, median and maximum scores. The lighter area indicates the distribution over the scores through a kernel-density estimation.

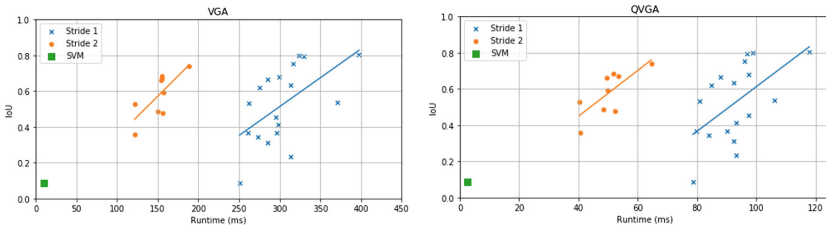


Fig. 4. IoU against runtime per image. The mean of the runtime is taken over 100 iterations. Results for networks with stride 1 and stride 2 are split and plotted with blue crosses and orange circles respectively. Additionally, for each group the linear fit to the data is shown. The baseline score of the SVM is marked as a green square. Left: Full VGA (640×480), right: QVGA (320×256); note the different timescales. (Color figure online)

From this figure we can conclude the same as from Fig. 3, that introducing a stride of 2 does not significantly reduce performance, but with the addition that it does make the network run significantly faster. The best network with stride 2, $L_3F_5M_2S_2$, has an IoU of only 0.066 less than the best network (a drop of just 8%), but runs over twice as fast, at more than 5 frames per second. On the lower resolution of 320×256 the best stride 2 networks achieve frame rates of 15 to 20 frames per second. Table 1 lists the results for all networks.

Table 1. Segmentation scores and runtimes obtained by networks

Layers	Filters	Mult	Stride	θ^*	IoU	Time (ms)	
						640 × 480	320 × 256
3	3	1.25	1	0.07	0.086	250	78
3	3	1.25	2	0.07	0.356	121	40
3	3	1.5	1	0.30	0.366	261	79
3	3	1.5	2	0.25	0.529	121	40
3	3	2	1	0.33	0.456	295	97
3	3	2	2	0.31	0.478	156	52
3	4	1.25	1	0.23	0.343	273	84
3	4	1.25	2	0.31	0.487	149	48
3	4	1.5	1	0.16	0.313	285	92
3	4	1.5	2	0.28	0.682	155	51
3	4	2	1	0.31	0.413	297	93
3	4	2	2	0.26	0.660	154	49
3	5	1.25	1	0.22	0.365	296	90
3	5	1.25	2	0.25	0.671	155	53
3	5	1.5	1	0.49	0.233	312	93
3	5	1.5	2	0.25	0.590	156	49
3	5	2	1	0.42	0.538	370	106
3	5	2	2	0.24	0.738	188	64
4	3	1.25	1	0.43	0.531	262	80
4	3	1.5	1	0.39	0.620	274	84
4	3	2	1	0.49	0.754	316	96
4	4	1.25	1	0.50	0.666	285	87
4	4	1.5	1	0.23	0.678	299	97
4	4	2	1	0.36	0.801	323	98
4	5	1.25	1	0.74	0.632	313	92
4	5	1.5	1	0.41	0.794	329	96
4	5	2	1	0.46	0.804	397	117
SVM					0.085	10	3

5.2 Multi-class Segmentation

Binary classification is too limited for robotic football, or other real world scenarios. To study the more general usability of our method, we extend the binary-class segmentation case from Sect. 5.1. The same dataset as before is used, but with additionally goalposts annotated as a third class. We selected the best stride 1 and best stride 2 networks to train. These two networks are kept the same, except for an additional channel added to the last decoding layer and to the softmax layer.

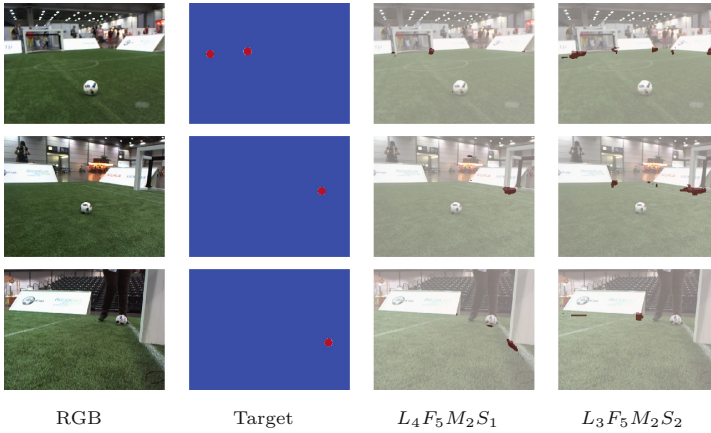


Fig. 5. Examples of input, target and segmentation outputs for goalposts for the two trained networks.

We found it to be difficult for the networks to successfully learn to segment the full goalpost, not being able to discern higher sections above the field edge from parts of the background. Combined with the fact that the robots typically use points on the ground, as they are able to better judge their distance, we select the bottom of the goalposts by labelling a circle with a radius of 20 pixels in the target output where the goalposts touch the field in the images.

Because of the additional difficulty of learning features for an extra class, the IoU score for the ball class dropped slightly for both networks: to 0.754 for $L_4F_5M_2S_1$ and to 0.708 for $L_3F_5M_2S_2$, compared to 0.804 and 0.738 in the binary task. The scores reached for the goalpost class were 0.273 and 0.102 respectively. Although this does not reach the same level as for the ball class, the networks are still able to mark out the bottom of the goal posts in the examples shown in Fig. 5. Because only some additional operations in the last layers, the run times are comparable to the binary equivalents: 414 and 196 ms.

The worse scores on the goalposts are mostly due to false positives, either marking too large an area or mislabelling other objects, especially in the case of the stride 2 network. Several reasons contribute to this. Firstly, the data used is typical for an image sequence of a robot playing a game of football, where it most of the time is focused on the ball. This results in goals being less often visible, and thus in the data being unbalanced: at least one ball is visible in all 1000 images, whereas at least one goal post is visible in only 408 images. Secondly, goal posts are less feature-rich, so more difficult to discern from other objects. Finally, our annotation method does not mark a well-distinguished area, making it harder for the networks to predict its exact shape. Further research is required to alleviate these issues and improve performance, however the results obtained here provide evidence that our approach can handle multi-class segmentation with only little performance reduction.

6 Conclusions

We have developed a minimal deep learning semantic segmentation architecture to be run on minimal hardware. We have shown that such a network can achieve good performance in segmenting the ball, and show promising results for additionally detecting goalposts, in a RoboCup environment with a useful frame rate. Table 2 lists the resolutions and frame rates reported by other authors alongside ours. It must be noted that a direct comparison is difficult, because of the different outputs of the used CNNs and the different robot platforms, but our approach is the only one that has all of the following properties:

- Processes full VGA images at 5 fps and QVGA at 15 to 20 fps⁴
- Can handle multiple image dimensions without retraining
- Does not require task specific knowledge to achieve high frame rate
- Achieves all this on minimal mobile hardware

For achieving full object localisation as given by the other solutions, additional steps are still required. However, because the output of the semantic segmentation is in the same form as a lookup table based labelling approach, any already existing methods built on top of such a method can directly be reused. For instance, an efficient—and still task agnostic—connected component based method previously developed by us readily fits onto the architecture outlined here and performs the final object detection step within only 1 to 2 ms.

Table 2. Reported resolutions and frame rate of DL application in RoboCup domain

	Resolution	FPS	Notes
Speck et al. [16]	200 × 150	3	Predict separate ball x and y
Albani et al. [1]	N/A	11–22	Task dependent region proposal
Cruz et al. [5]	24 × 24	440	Task dependent region proposal
Javadi et al. [10]	N/A	240	No loss: 6 fps; task dependent
Da Silva et al. [6]	110 × 110	8	Predict end-to-end desired action
Hess et al. [8]	32 × 32	50	Focus on generation of training data
Schnekenburger et al. [14]	640 × 512	111	GTX-760 GPU; 19 fps on i7 CPU
Ours	640 × 480	5	$L_3F_5M_2S_2$
	320 × 256	15	$L_3F_5M_2S_2$; $L_3F_4M_{1.5}S_2$: 20 fps

By delaying the use of task dependent methods, one actually has an opportunity to optimise the segmentation output for such methods, by varying the threshold used to determine the final class pixels. For specific use cases it may be desirable to choose a threshold that represents a preference for either high true positive rate (recall), e.g. when a robot’s vision system requires complete

⁴ Actually a resolution slightly larger than QVGA is used, as the stride 2 networks require dimensions that are multiples of 32.

segmentation of the ball and/or it has good false-positive filtering algorithms, or for low false positive rate (fall-out), e.g., when it can work well with only partly segmented balls, but struggles with too many false positives.

References

1. Albani, D., Youssef, A., Suriani, V., Nardi, D., Bloisi, D.D.: A deep learning approach for object recognition with NAO soccer robots. In: Behnke, S., Sheh, R., Sarel, S., Lee, D.D. (eds.) RoboCup 2016. LNCS (LNAI), vol. 9776, pp. 392–403. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68792-6_33
2. Allali, J., et al.: Rhoban football club: RoboCup humanoid kid-size 2016 champion team paper. In: Behnke, S., Sheh, R., Sarel, S., Lee, D.D. (eds.) RoboCup 2016. LNCS (LNAI), vol. 9776, pp. 491–502. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68792-6_41
3. Badrinarayanan, V., Kendall, A., Cipolla, R.: SegNet: a deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **39**(12), 2481–2495 (2017)
4. Cheng, G., Zelinsky, A.: Real-time vision processing for a soccer playing mobile robot. In: Kitano, H. (ed.) RoboCup 1997. LNCS, vol. 1395, pp. 144–155. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-64473-3_56
5. Cruz, N., Lobos-Tsunekawa, K., Ruiz-del-Solar, J.: Using convolutional neural networks in robots with limited computational resources: detecting NAO robots while playing soccer. In: Akiyama, H., Obst, O., Sammut, C., Tonidandel, F. (eds.) RoboCup 2017. LNCS (LNAI), vol. 11175, pp. 19–30. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00308-1_2
6. Da Silva, I.J., Vilao, C.O., Costa, A.H., Bianchi, R.A.: Towards robotic cognition using deep neural network applied in a goalkeeper robot. In: 2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR), pp. 1–6. IEEE (2017)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
8. 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. LNCS (LNAI), vol. 11175, pp. 33–44. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00308-1_3
9. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv preprint [arXiv:1704.04861](https://arxiv.org/abs/1704.04861) (2017)
10. Javadi, M., Azar, S.M., Azami, S., Ghidary, S.S., Sadeghnejad, S., Baltés, J.: Humanoid robot detection using deep learning: a speed-accuracy tradeoff. In: Akiyama, H., Obst, O., Sammut, C., Tonidandel, F. (eds.) RoboCup 2017. LNCS (LNAI), vol. 11175, pp. 338–349. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00308-1_28
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436 (2015)
12. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: imagenet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 525–542. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32

13. Ronneberger, O., Fischer, P., Brox, T.: U-Net: convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) MICCAI 2015. LNCS, vol. 9351, pp. 234–241. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24574-4_28
14. Schnekenburger, F., Scharffenberg, M., Wülker, M., Hochberg, U., Dorer, K.: Detection and localization of features on a soccer field with feedforward fully convolutional neural networks (FCNN) for the adult-size humanoid robot sweaty. In: Proceedings of the 12th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Birmingham (2017)
15. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. *Nature* **529**(7587), 484–489 (2016)
16. Speck, D., Barros, P., Weber, C., Wermter, S.: Ball localization for robocup soccer using convolutional neural networks. In: Behnke, S., Sheh, R., Sarel, S., Lee, D.D. (eds.) RoboCup 2016. LNCS (LNAI), vol. 9776, pp. 19–30. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68792-6_2
17. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint [arXiv:1606.06160](https://arxiv.org/abs/1606.06160) (2016)