Real-time Scene Understanding using Deep Neural Networks for RoboCup SPL

Marton Szemenyei,[1] Vladimir Estivill-Castro[2]

[1] Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Budapest, Hungary
[2] Institute for Intelligent and Integrated Systems, Griffith University, Brisbane, QLD, Australia

Abstract
Convolutional neural networks (CNNs) are the state-of-the-art method for most computer vision tasks. However, their excessive computational requirements make the deployment of CNNs on mobile or embedded platforms challenging. In this paper, we present and end-to-end neural network solution to scene understanding in the context of robot soccer. Our system uses two key neural networks: one to perform semantic segmentation on an image, and another to propagate class labels between consecutive frames. Our networks are trained on synthetic datasets and fine-tuned on a set consisting of real images taken using a Nao robot. Furthermore, we discuss and evaluate several practical methods for increasing the efficiency and performance of our networks. Finally, we present NaoDNN, a C++ neural network library designed for fast inference on the Nao robots.

1. Introduction

Deep learning [1] has been one of the most rapidly advancing fields of computer science in the last decade. While deep neural networks (DNNs) have a vast area of applications, they are undoubtedly the most popular in the field of intelligent perception, especially computer vision. While the focus of the field has been on image classification [2], in the last few years there have been considerable advances on using DNNs on other vision tasks, such as object detection [3], tracking [4] and segmentation [5].

Due to the rapidly increasing power of hardware, the usually computationally expensive networks have started to appear in mobile and embedded systems [6]. Several teams [7–11] have used convolutional neural networks in the 2017 SPL league in RoboCup to classify relevant objects on the soccer field. However, due to the limitations of the robot's hardware, these networks were relatively shallow and were designed to classify fixed-resolution image patches only. This meant that the teams had to use separate object proposal methods to feed their network with candidate image regions.

In this paper we propose an end-to-end real-time object detection method for the Nao robots using deep neural networks. Our method combines two separate networks to achieve high accuracy at reasonable speed. The first network is a deep neural network trained to perform semantic segmentation (pixel-wise classification) on the image from the robot's camera. The second is a somewhat smaller network, trained to propagate the class labels from the previous image onto the next. Our system is capable of localizing four foreground classes (ball, robot, goalpost, and field line), without the use of a separate object proposal system.

Furthermore, we also present our experiments with methods used to accelerate deep neural networks on the Nao robots. These techniques include using field edge detection to reduce the image size without decreasing the resolution of the relevant image parts, and pruning the weights of the neural network. We also experiment with different network structures in order to find the optimal accuracy/runtime trade-off.

Finally, we present NaoDNN, a lightweight C++ deep neural network library. NaoDNN is a forward-only library, designed for maximum performance on the Nao robot hardware. The library has no dependencies, and does not use the C++11 standard, which makes it easy to compile for the robots. Moreover, it has been designed to compile using the strictest compiler settings. Our implementation also offers compatibility with the popular Pytorch [12] DNN framework, with the ability to import neural nets trained using Pytorch. The NaoDNN library, the code and datasets used to train our networks are available on our website (`mipal.net.au`).

2. Related Work

Deep neural networks have become a widely-used and powerful solution to numerous machine learning problems. The renewed interest in DNNs was largely sparked by the dramatic increase in the availability of high quality datasets and computational resources. The lack of these was one of the major barriers to training deep neural networks, in addition to some numerical problems [13]. Their applications are countless, ranging from natural language processing problems such as query answering [14] and translation [15] to implementing Turing machines [16] or other complex recurrent systems. They are also frequently used as agents in games, trained using some form of reinforcement learning with considerable success, as demonstrated by AlphaGo Zero [17].

The use of deep learning is perhaps most prominent in the field of computer vision, where deep neural networks are used for standard classification [2], object detection [3] and (semantic or instance) segmentation tasks [5, 18]. Lately, the applications expanded to several more complex areas, such as video and motion analysis [4], image captioning [19], translation and generation [20].

Semantic segmentation is one of the major tasks aiming to achieve visual scene understanding. The objective is to segment the image by classifying each pixel individually. The simplest way to achieve this with neural networks is to use a classification network, and replace the final fully connected layers by (usually 1x1) convolutional layers. This network can output a segmented image at a lower resolution, which can be upsampled using techniques, such as bilinear upsampling [21]. Other works [22] employ a superpixel segmentation method, and classify these superpixels individually in order to approximate object boundaries with higher accuracy.

One of the first major works in this field proposed the SegNet architecture [21], which uses so-called unpooling layers to upsample the feature maps. In SegNet the max pooling layers in the first half of the network store the index of the maximum value and share this information with the corresponding unpooling layer. This extra piec of information allows the unpooling layer to recover the spatial information lost during downscaling. Nonetheless, the full feature map is not recovered, since the non-maximum values are permanently lost.

In the last few years, several important advances have been made to improve the accuracy of SegNet-based segmentation, especially when it comes to capturing the fine details of objects. The first of these improvements is the Fully Convolutional Network (FCN) architecture [5], which introduced shortcuts (skip connections) from the front layers of the network. By adding shortcuts from early layers, the final layer has more information on the fine-resolution details of the image, resulting in better approximation of object boundaries. Shortcuts also improve the convergence properties of deep neural networks considerably [23].

Choosing the upscaling and downscaling methods in the network can also affect the performance significantly. The FCN network uses strided convolutions instead of pooling and using transposed convolutional layers instead of unpooling in order to implement a learnable downsampling and upsampling operations, increases the complexity of network greatly [5]. Wang et al. [24] introduced the dense upsampling convolution operation (DUC), which was shown to increase the accuracy further, although at a considerable increase of computational cost.

The field of view of the final classification neuron is another important property that influences the accuracy of the segmentation network, since it determines the amount of contextual information the final neuron can use to determine the class of each pixel. Chen et. al. [25] showed that using dilated/atrous convolutional filters increases the performance without increasing the computational cost. Pooling operations may also be atrous [25], resulting in a similar improvement.

While convolutional neural networks have achieved staggering accuracy in numerous applications, their power comes with high computational cost. Such high computational cost seems to prohibit the use of CNNs on board of mobile and embedded platforms, but recent research in CNN optimisation is alleviating this issue. The focus of this field to reduce the size of trained models as well as the computation needed for inference. Several methods exist for this purpose, such as weight sharing [26], quantization [6] and low-rank approximation [27]. Considerable research has been devoted to binary networks [28].

The technique most relevant to our application is called pruning [29]. During the pruning process, a ranking method is used to order the weights or neurons of a layer by importance, then a fixed percentage of the least important neurons/connections are deleted/set to zero. The network is fine-tuned afterwards, while keeping the pruned elements on zero value. Several ranking methods exist ranging from brute-force methods, that simply use the magnitude of the weights to more complex ones, such as pruning weights so that the change in the network's loss function would be minimal [30].

2.1. Computer Vision in RoboCup

The stated goal of RoboCup 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 21st 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 vision pipelines used by the competing teams have changed in tandem, going from human-engineered vision methods [31, 32] to pipelines relying increasingly on machine learning. Several teams have used convolutional neural networks either for binary classification tasks [7, 8] or to detect several relevant object categories [9, 10]. 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 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.

One of the most important advances of recent years is the work published by Hess et al. [11] in which they present a high-quality virtual RoboCup environment created in Unreal Engine. Using their work allows one to easily create large datasets of realistic images of a soccer field along with pixel-level semantic labeling. Since the performance of a trained neural network is highly dependent on the quality and quantity of the training data, and creating a large hand-labeled database is highly time-consuming, their work was profoundly valuable for our research.

3. Preparation of the Training Data

In most machine learning applications the quality and amount of the training data is a major determinant of the algorithm's eventual performance. For training the semantic segmentation network, we created a synthetic image set of 5000 images using the tool presented by Hess et al [11]. We used 100 different random sets of environmental variables, and generated 50 images with each setting. The images were separated into train and test sets randomly, using an 80-20 division. The automatically generated labels are available as PNG images, and contain labels for all five relevant categories (background, ball, robot, goal and line).

In addition, we modified the project blueprint to allow for the creation of image sequences instead of images of independent, randomly-arranged scenes. We used this mode to create a dataset of 800 images to train the label propagation network. This database also features 100 individual image sequences with different random scene parameters. In each sequence, however, the position and orientation of the camera and field objects only changes marginally between consecutive frames.

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 as a result of the pre-training, a much smaller dataset is sufficient than would be otherwise required. For these reasons, we created a real semantic segmentation database consisting of 570 images taken at 3 separate locations: at the venue of RoboCup17, at the venue of IJCAI17 and in our lab at Griffith University. A portion of this database consists of image sequences, which are used as a dataset for label propagation.

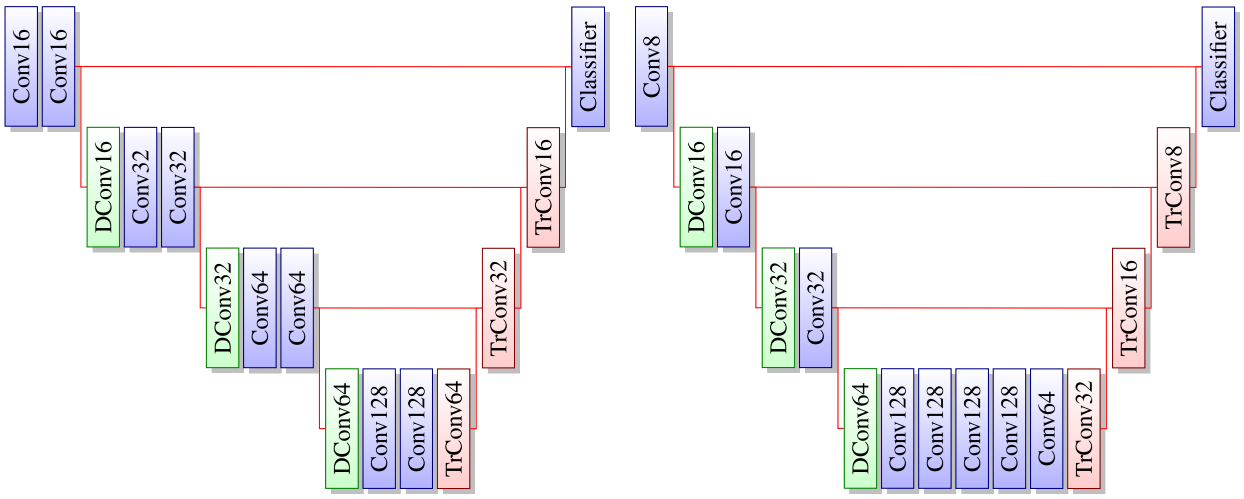
We manually annotated the images using a tool of our own creation. Our tool provides several ways to aid the annotation process, such as tools for drawing polygons and lines, as well as square and circular brush tools. The program also uses the superpixel segmentation method proposed by Li and Chen [33] 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.

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 aggressive, unique data augmentation in addition to standard techniques, such as flipping images horizontally. In order to emulate changes in lightning conditions, we applied random changes in the brightness and contrast of the images. To introduce further variation into the dataset, we also applied random shifts to the hue and saturation of our pictures, which may help the robots with unique carpet colors. In Section 6 we show that our data augmentation techniques improve the accuracy of the trained models greatly.

4. Model Selection and Training

For training the network we use the popular Pytorch framework [12] and perform the optimization using stochastic gradient descent (SGD) with momentum and weight decay regularization. During training, we used an adaptive learning rate schedule, which reduced the learning rate of the network after N consecutive epochs in which the validation loss could not fall below the current lowest value. We made a slight modification to Pytorch's learning rate scheduler, to allow us to reload the current best model in every learning rate reduction event. We observed that this made it more likely for the optimizer to find a new optimum after reducing the learning rate. We present the hyperparameters used for pre-training and fine-tuning the semantic segmentation and label propagation networks in Table 1.

One of the major challenges faced during training was due to the nature of the scene setup specific to robot soccer fields. In a usual image taken of the soccer field the vast majority of the pixels belong to the background class. In our datasets the ratio of background pixels was around 93-94%,



(a) Standard FCN-based

(b) "Pot-Bellied" FCN

Figure 1: Our model architectures. Blue nodes represent standard convolutional layers, green nodes are strided convolution layers, while red nodes are strided transposed convolution. Every level lowers the resolution by a factor of 2. Note, that batch normalization layers present after every convolution layer were omitted for brevity.

moreover, the rest distributed rather unevenly amongst the other classes. This results in a heavily unbalanced dataset, which can make convergence difficult, and may result in a final network that is heavily biased towards making false negative-type errors. It is important to note, that this imbalance stems from the distribution of classes *in the individual images themselves*, therefore it is not possible to re-sample the training set.

We used two solutions in order to combat this difficulty. First, we selected the images to be used in the training set so that they would contain a relatively high percentage of foreground pixels. Second, we used a weighted version of the 2 dimensional negative log likelihood (NLL) loss function, which is implemented in Pytorch, encouraging the network to emphasize the relevant object categories more.

Another important aspect of the training procedure was to find an efficient and powerful network structure. At first, we used an architecture based on FCN [5], using strided convolution for downsampling and transposed convolution for upsampling, as well as employing dilated convolutions to increase the field of view of the final classification layer. We avoided using DUC for upsampling, due its higher computational requirements. Our network had three modules consisting of convolutional and downsampling layers, combined with three upsampling layers, as illustrated in Figure 1a.

Most CNNs used for semantic segmentation are relatively waist-heavy, meaning that the middle section of 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 downsample the feature map quickly, then using a larger number of deep convolutional layers at the lowest level, followed by a similarly shallow and quick upsampling. In Section 6 we demonstrate that this network structure is much more efficient, providing better accuracy for lower computational cost. Figure 1b illustrates our new alternative, the "Pot-Bellied" (PB-FCN) architecture.

Our training procedure consists of three steps. First, the first half of the segmentation network is trained on the dataset provided by Hess et. al [11] for classification. We modified the dataset by separating the background a field line classes. Next, the full segmentation network is trained on the synthetic database. Finally, the full segmentation network is finetuned on the real database. The training procedure for the label propagation network is similar, except the the first step is ommitted.

5. Real-time Implementation

In order to ensure real-time performance of our object detection pipeline, we had to employ several techniques to improve the speed of the trained neural network. In this case, these improvements were essential, since the networks presented in the previous sections took approximately 1 second to run on a Nao V5 robots using the Darknet library [34]. For all our experiments, we used 160x120 images in YUV color space.

The first technique we employed was weight pruning: since convolution is implemented as a matrix multiplication, setting weights to zero increases the efficiency signifi-

Training	Segm.	Finetune	LP	Finetune
Learning Rate	0.1	0.01	0.2	0.05
Momentum	0.5	0.1	0.5	0.1
Weight Decay	1e-3	1e-3	1e-5	1e-5
LR Reduction	0.5	0.5	0.5	0.5
Patience (N)	20	50	10	25
Batch Size	32	8	16	8
Epochs	200	500	100	250

Table 1: Hyperparameters used for the training procedures.

cantly, even when an extra operation (checking if the weights are zero) is introduced. We used a brute-force pruning technique, simply setting approximately 75% of the weights in every layer to zero, and then fine-tuning the network, while forcing the pruned weights to remain zero. We found, that this technique reduced the runtime of the network by approximately 70%.

Moreover, our vision pipeline includes a hand crafted field detection system, which is used by our network to crop the part of the image, outside the field (this is usually the top part of the image), and only run the network on the relevant part. 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 may avoid learning complex backgrounds outside the field (which are easily confused with field objects). While this technique provides considerable improvement in the networks speed, this improvement is highly dependent on the robot's position in the field. For this reason, we used uncropped images when comparing the execution times of different models and methods.

5.1. Label Propagation

Our other major attempt at increasing the speed of our pipeline was to employ label propagation, a technique to estimate the labels of the next image by using the labels of the previous one. By only running the main neural network every 10 or 20 frames, we are able to achieve a considerable increase in speed, provided that accurate label propagation can be implemented using a significantly faster algorithm.

In our first experiments, we employed Gunnar Farneback's dense optical flow algorithm [35] to move the labels to their new location. While the algorithm's speed was more than satisfactory, there were issues with the accuracy. Namely, small, single-pixel errors would accumulate over time, constantly eroding small objects. Also, the optical flow-based method is completely unable to handle faster movements or new objects appearing in the image (or partially seen objects sliding in).

Our second choice was to implement label propagation with a considerably smaller version of PB-FCN (Figure 1), which would run at approximately twice the speed of the segmentation network. This network uses only the Y channels of the YUV images, and it also takes the difference of the two frames as an input. The labels of the previous image are concatenated to this array, resulting in an 8-channel input. For numerical reasons, the binary labels were scaled between -1 and 1. The label propagation network was trained using the synthetic and real datasets mentioned in Section 3. We used the same data augmentation techniques, and trained the network to be able to predict the new labels in both ways (previous-to-next and vice versa), since the vast majority of movements might occur in both directions.

Since this method is able to combine knowledge about the visual appearance of the classes and the movement between the images it is arguably able to account for the appearance of new objects and handle larger movements. Moreover, for the same reason, the label propagation network has some form of self-correcting ability, thus misclassifying a pixel in one frame does not mean that the error will be carried on until the next run of the segmentation network. We observed during our experiments that the label propagation network seemed to be able to incrementally correct the mistakes made by the segmentation network, especially when the robot was not moving (Figure 2).

Layer	**Optimization**
Convolutional	Pruning, Crop
Transposed Conv	Pruning, Crop
Fully Connected	Pruning, Crop
Pooling (Avg/Max)	Crop
Activations	In-place, Crop
Batch Normalization	In-place, Crop Pre-compute $\frac{1}{\sqrt{var}}$
Concatenation	Crop
Shortcut	In-place, Crop
ReOrg/Pixel Shuffle	Crop
SoftMax	Crop

Table 2: List of layers implemented in the NaoDNN library, along with the optimization techniques we applied.

Technique	**TPA**	**MCA**	**MIoU**
Baseline	97.72	92.19	75.57
Data Augmentation	0.61	2.55	6.66
Field Extraction	0.31	0.82	2.36
Reload	0.09	1.08	0.19
Prune	-0.18	-0.8	-0.4

Table 3: Comparison of the training techniques we used to increase the accuracy of out networks. We present the changes compared to the baseline.

5.2. The NaoDNN Library

The last important detail of implementation is finding an appropriate neural network library to use on the Nao robots. The first option is Caffe [36], which is a relatively old library with numerous dependencies, making it difficult to compile for the Nao robot. The other option is Darknet [34], which has no dependencies and is more recent. Darknet, however, lacks support for several important features we used in our network, such as dilated convolutions and affine batch normalization.

For this reason, we created our own library called NaoDNN, which is heavily based on Darknet, and uses C++ to implement some of 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.

We have updated the library's code to make it compatible with Pytorch, which included adding support for dilated convolutions, output padding for transposed convolutions, and affine batch normalization layers. Altogether, NaoDNN is fully compatible with neural networks trained in Pytorch, and we provide code to export the weights 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. Table 2 provides al list of the layers we implemented and the optimisation we applied.

6. Experimental Results

In this section we present the result of our experiments with the trained networks. In the first part, we focus on testing the accuracy of dirrenet models and techniques employed. Our experiments regarding the speed of our pipeline on the Nao V5 robots are presented in the second part of this section. Some of the best and worst results of the segmentation are displayed in Figure 3.

Model	**TPA**	**MCA**	**MIoU**
FCN	98.42	94.95	80.31
PB-FCN	98.50	94.40	81.30
PB-FCN-VGA	98.64	**95.74**	81.09
Resnet152	**98.71**	94.88	**83.98**

Table 4: Comparison of the different neural netowrk architectures.

6.1. Tests on Accuracy

For comparing the accuracy of different models and techniques, we use three measures. The first is the Total Pixel Accuracy (TPA), which is simply the percentage of pixels classified correctly. The second is the Mean Class Accuracy (MCA), which is TPA computed separately for each class, and averaged. With our unbalanced dataset there is significant trade-off between these two measures: While TPA will tend to be higher with models that are more likely to err on the side of background, MCA will be higher for models that are more likely to make false positive predictions. The third measure is Mean Intersection over Union (MIoU), which (on our dataset at least) prefers models with minimal confusion between foreground classes.

In our first experiment, we demonstrate the increase in accuracy provided by techniques such as our data augmentation operations, field extraction, reloading learning rate scheduler, and the effect pruning. We measured the improvements of these techniques separately on a fine-tuned PB-FCN network, therefore the improvements we present in Table 3 are not additive. The results indicate that our techniques increase the accuracy considerably, while our brute-force pruning retains most of the networks predictive power.

Secondly, we compare the results of different models. In this comparison we use all of our aforementioned techniques. In Table 4, we compare four different models: The standard FCN-based model, our PB-FCN used on both 160x120 and 640x480 resolution images, and an extremely deep segmentation network using Resnet152 [23] and deep DUC upsampling. From these result, we can draw several conclusions: First, PB-FCN is slightly more powerful compared to the standard FCN structure. Second, a comparatively shallow PB-FCN loses surprisingly little in terms of accuracy compared to the Resnet152-based model, especially when used on VGA resolution.

Lastly, we compare the results of our pruned, fine-tuned PB-FCN-based label propagation network with the results acquired using optical flow. The results in Table 5 indicate that neural label propagation clearly outperforms the optical-flow based method. Note, that optical flow producing slightly better MIoU results is an artifact caused by the heavily unbalanced dataset (optical flow introduces minimal confusion between foreground classes). We used the real image dataset for all the results in this section.

6.2. Comparison against other solutions

We also compare our results against the networks or detection pipelines used by other teams. First, we evaluate the first half of our PB-FCN network against the classification results reported by Hess et al. [11] using the same test dataset. Our algorithm clearly outperforms theirs, producing XY.xy% accuracy compared to 94.4% using a synthetic dataset of matching size.

We also compared our full segmentation network against UT Aston Villa's detection pipeline using our own validation set consisting of real images. In this test we simply counted the amount of correct detections, as well as false positives and false negatives. The results in Table 6 show that our method seems to achieve considerably higher accuracy.

6.3. Tests on Execution Time

We have also tested the execution time of our methods on a Nao V5 robot. Table 7 shows the execution time of the pruned versions of the models compared in the previous subsection. For reference, we also included the non-pruned version of PB-FCN. The results show a clear improvement as a result of pruning, and that PB-FCN outperforms the vanilla FCN in speed as well. We remark that the data shows that running a relatively shallow network on a higher resolution image seems to be much faster than running ResNet on a downscaled version, while providing comparable accuracy.

In Table 7 we also present the comparison of label propagation using optical flow and CNNs. The results show that the extra accuracy coming with the neural network comes at lower speeds. Still, the fully neural vision pipeline is able to run at 7 frames per second, which is sufficient. For reference, we include our measurements of the speed of the UT Aston Villa team's vision pipeline. The comparison shows, that although we were able to achieve significant improvements in accuracy and the neural network's efficiency, it is still several times slower than other methods.

Model	TPA	MCA	MIoU
Optical Flow	95.82	86.15	82.7
PB-FCN	96.52	90.7	79.15

Table 5: Comparison of the two methods we used for Label Propagation.

Method	Accuracy	False positives	False negatives
UT AV	BAD	BAD	BAD
PB-FCN	GOOD	GOOD	GOOD

Table 6: Comparison of PB-FCN agains UT Aston Villa's detection pipeline.

7. Conclusion

In this paper, we presented a deep neural network-based method for scene understanding in the context of robot soccer. Our method uses a semantic segmentation network and a separate label propagation net to increase the frame rate of the vision system. With our experiments, we demonstrated the efficiency of our method, including the improvements we achieved using our data augmentation techniques, pruning and field-edge cropping. Our method has superb accuracy at satisfactory speed.

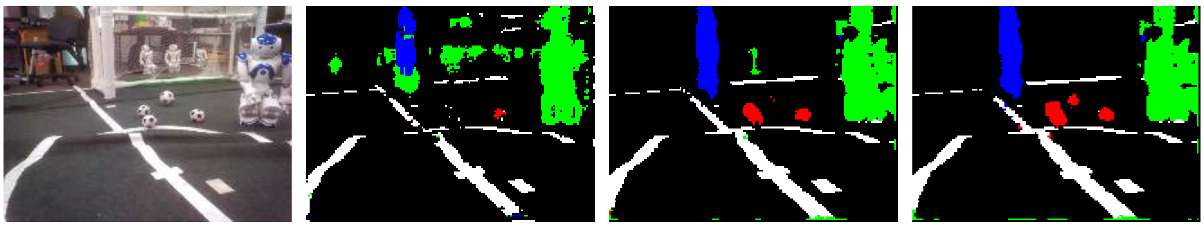
We also presented large semantic segmentation and label propagation datasets consisting of synthetic images, as well as small real datasets for the same tasks, including a tool for manual pixel-wise labeling of images. Finally, we presented a Pytorch-compatible C++ deep neural network library designed for fast inference on the Nao robots supporting the acceleration techniques discussed in this paper. Our library has been designed to compile for the Nao robots using the strictest compiler settings.

Acknowledgments

Our research has been supported by NVIDIA, who kindly donated a TitanX Pascal GPU for training neural networks.

Model	Execution Time (ms)	FpS
FCN	520	0.5
PB-FCN	1160	0.9
PB-FCN Pruned	290	3.4
PB-FCN-VGA	2,500	0.4
Resnet152	8,000	0.125
Optical Flow	80	12.5
Neural LP	130	7.6
UT AV	40	25

Table 7: Comparison of the execution times of different models.



(a) Original image (b) Semantic segmentation (c) Label propagation after 5 frames (d) Ater 10 frames

Figure 2: The self-correcting ability of the label propagation network.

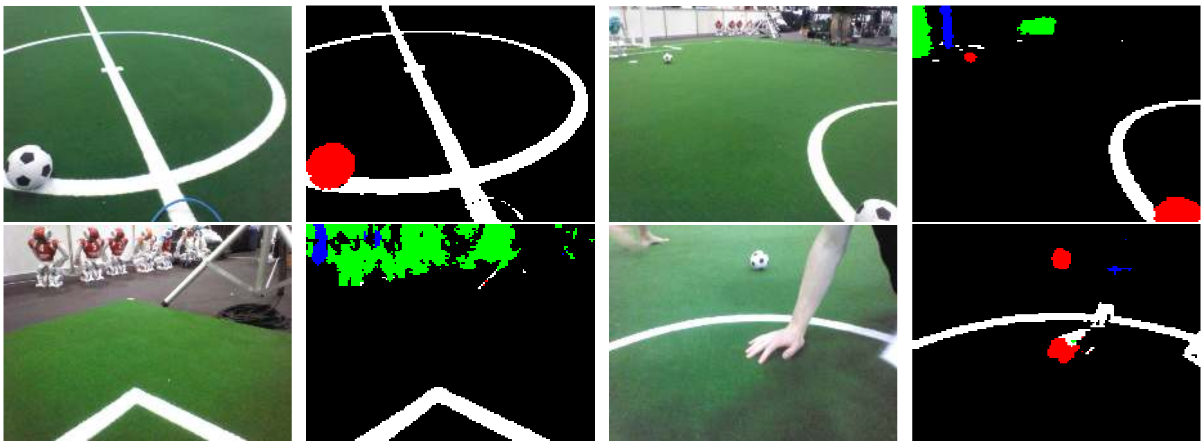


Figure 3: A few examples of good (top) and bad (bottom) results.

Our research collaboration is a part of the Erasmus Mundus PANTHER programme.

References

1. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
2. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” in *Proceedings of the IEEE*, 1998, pp. 2278 – 2324.
3. S. Ren, K. He, R. Girshick, and J. Sun, “Rfaster r-cnn: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.
4. P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. HazÄśrbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, “Flownet: Learning optical flow with convolutional networks,” *CoRR*, 2015.
5. E. Shelhamer, J. Long, and T. Darrell, “Fully convolutional networks for semantic segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, 2017.
6. J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, “Quantized convolutional neural networks for mobile devices,” in *CVPR*, 2016.
7. S. M. A. Mohammad Javadi and, S. Azami, S. S. Ghidary, S. Sadeghnejad, and J. Baltes, “Humanoid robot detection using deep learning: A speed-accuracy tradeoff,” in *RoboCup Symposium*, 2017.
8. N. Cruz, K. Lobos-Tsunekawa, and J. R. del Solar, “Using convolutional neural networks in robots with limited computational resources: Detecting nao robots while playing soccer,” in *RoboCup Symposium*, 2017.
9. S. O’Keeffe and R. Villing, “A benchmark data set and evaluation of deep learning architectures for ball detection in the robocup spl,” in *RoboCup Symposium*, 2017.
10. J. Menashe, J. Kelle, K. Genter, J. Hanna, E. Liebman, S. Narvekar, R. Zhang, and P. Stone, “Fast and precise black and white ball detection for robocup soccer,” in *RoboCup Symposium*, 2017.
11. T. Hess, M. Mund, T. Weis, and V. Ramesh, “Large-scale stochastic scene generation and semantic annotation for deep convolutional neural network training in the robocup spl,” in *RoboCup Symposium*, 2017.
12. [Online]. Available: htpp://www.pytorch.org

13. X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

14. J. Yin, X. Jiang, Z. Lu, L. Shang, H. Li, and X. Li, “Neural generative question answering,” in *IJCAI*, 2016.

15. I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *27th International Conference on Neural Information Processing Systems*, 2014.

16. A. Graves, G. Wayne, and I. Danihelka, “Neural turing machines,” *CoRR*, 2014.

17. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, pp. 354–359, 2017.

18. O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015, pp. 234–241.

19. A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” in *CVPR*, 2015.

20. P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial nets,” in *CVPR*, 2017.

21. V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation.” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

22. F. Z. Xing, E. Cambria, W.-B. Huang, and Y. Xu, “Weakly supervised semantic segmentation with superpixel embedding,” in *IEEE International Conference on Image Processing*, 2016.

23. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016.

24. P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. W. Cottrell, “Understanding convolution for semantic segmentation,” *CoRR*, 2017.

25. L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *CoRR*, 2016.

26. S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” in *ICLR*, 2016.

27. A. S. Davis and I. Arel, “Low-rank approximations for conditional feedforward computation in deep neural networks,” *CoRR*, 2013.

28. M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -,” *CoRR*, 2016.

29. S. Anwar, K. Hwang, and W. Sung, “Structured pruning of deep convolutional neural networks,” *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, no. 3, 2017.

30. P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” in *ICLR*, 2017.

31. M. Schwarz, I snd Hofmann, O. Urbann, and S. Tasse, “A robust and calibration- free vision system for humanoid soccer robots,” in *RoboCup Symposium*, 2015.

32. S. Metzler, M. Nieuwenhuisen, and S. Behnke, “Learning visual obstacle detec- tion using color histogram features,” in *RoboCup Symposium*, 2011.

33. Z. Li and J. Chen, “Superpixel segmentation using linear spectral clustering,” in *CVPR*, 2015.

34. J. Redmon, “Darknet: Open source neural networks in c,” http://pjreddie.com/darknet/, 2013–2016.

35. G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” in *Scandinavian Conference on Image Analysis*, 2003, pp. 363–370.

36. [Online]. Available: http://caffe.berkeleyvision.org