

Neural Networks for Segmenting Neuronal Structures in EM Stacks

Dan Cireşan, Alessandro Giusti, Luca Gambardella, and Jürgen Schmidhuber

The Swiss AI Lab IDSIA (Dalle Molle Institute for Artificial Intelligence)
USI, SUPSI, Lugano, Switzerland
{dan,alessandro,luca,juergen}@idsia.ch,
WWW: <http://www.idsia.ch/~cirezan>

Abstract. A pixel classifier is driving our winning approach to neural structure segmentation in electron microscopy images. Without explicit feature computation, the label of each pixel (membrane or non-membrane) is predicted from raw pixel values in a square window centered on it. The classifier is a special type of feed-forward neural network trained by plain gradient descent. The input layer maps each window pixel to a neuron. It is followed by a succession of convolutional and max-pooling layers which preserve 2D information and extract features with increasing levels of abstraction. The output layer produces a calibrated probability for each class, and is subjected to very mild post-processing. The approach outperformed all other entries of the competition in all considered metrics: values for *rand error*, *warping error* and *pixel error* were $48 \cdot 10^{-3}$, $434 \cdot 10^{-6}$ and $60 \cdot 10^{-3}$, respectively.

1 Introduction

We solve the problem of supervised segmentation of neural membranes in EM stacks by means of a Deep Neural Network (DNN) used as a pixel classifier; the network computes the probability of a pixel being a membrane, using as input the image intensities in a square window centered on the pixel itself. An image is then segmented by classifying all of its pixels, and very mild postprocessing.

Our DNNs are inspired by convolutional neural networks introduced in 1980 [10], improved in the 1990s [14], refined and simplified in the 2000s [2, 19], and brought to their full potential by making them both wide and deep [6]. Lately, DNN proved their mettle on data sets ranging from handwritten digits (MNIST) [4, 6], handwritten characters [5] to 3D toys (NORB) and faces [21]. Training huge nets used to require months or even years on CPUs, where high data transfer latency prevented multi-threading code from saving the situation. A fast GPU implementation [4, 6] overcomes this problem, speeding up plain CPU code by up to two orders of magnitude.

Many other approaches to image segmentation rely on learned classifiers. In most binary segmentation problems, classifiers are used to compute one or both of the following probabilities: (a) probability of a pixel belonging to each class; (b) probability of a boundary dividing two adjacent pixels. Segmentation

through graph cuts [3] uses (a) as the unary term, and (b) as the binary term. Some use an additional term to account for the expected geometry of neuron membranes[13].

In our approach, we compute pixel probabilities only (point (a) above), and directly obtain a segmentation by mild smoothing and thresholding, without using graph cuts. Our main contribution lies therefore in the classifier itself. Others have used off-the-shelf random forest classifiers to compute unary terms of neuron membranes [12], or SVMs to compute both unary and binary terms for segmenting mitochondria [15]. The former approach uses haar-like features and texture histograms computed on a small region around the pixel of interest, whereas the latter uses sophisticated rotational [11] and ray [20] features computed on superpixels [1]. Feature selection mirrors the researcher’s expectation of which characteristics of the image are relevant for classification, and has a large impact on classification accuracy. In our approach, we bypass such problems, using raw pixel values as inputs. Due to their convolutional structure, the first layers of our network automatically learn to compute meaningful features during training.

2 Methods

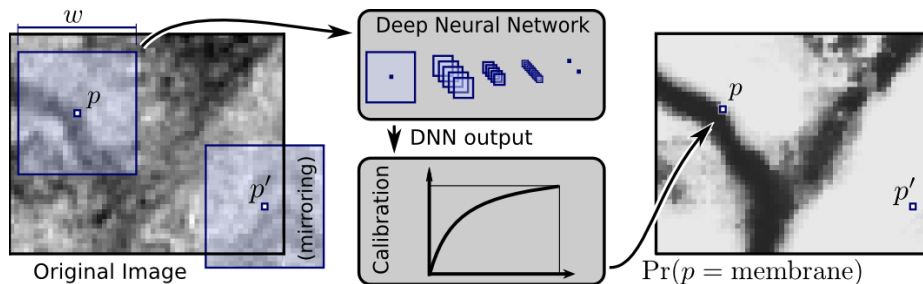


Fig. 1: Overview of our approach (see text).

For each pixel we consider two possible classes, *membrane* and *non-membrane*. Our DNN classifier (Section 2.1) computes the probability of a pixel p being of the former class, using as input the raw intensity values of a square window centered on p and with an edge of w pixels – w being an odd number in order to keep symmetry. When a pixel is close to the edge of the image, its window will include pixels outside the image boundaries; such pixels are synthesized by mirror-reflecting pixels in the actual image across the boundary (see Figure 1).

The classifier is first trained using the provided training images (Section 2.2). After training, to segment a test image, the classifier is applied to all its pixels, thus generating a map of *membrane* probabilities – i.e. a new real-valued image the size of the input image. A binary segmentation of the membrane is

obtained by mild postprocessing techniques discussed in Section 2.3, followed by thresholding.

2.1 DNN architecture

A DNN [7] consists of a succession of convolutional and max-pooling layers. It is a general, hierarchical feature extractor that maps raw pixel intensities of the input image into a feature vector to be classified by several fully connected layers. All adjustable parameters are jointly optimized through minimization of the misclassification error over the training set.

Each **convolutional layer** performs a 2D convolution of its input maps with a square filter. The resulting activations of the output maps are given by the sum of the convolutional responses which are passed through a nonlinear activation function.

The biggest architectural difference between our DNN and earlier CNN [14] is the use of **max-pooling layers** [16, 18, 17] instead of sub-sampling layers. Also, our DNN have many more maps per layer, and thus many more connections and weights. The output of a max-pooling layer is given by the maximum activation over non-overlapping rectangular regions. Max-pooling creates slight position invariance over larger local regions and down-samples the input image.

After 1 to 4 stages of convolutional and max-pooling layers several **fully connected layers** further combine the outputs into a 1D feature vector. The output layer is always a fully connected layer with one neuron per class (two in our case). Using a softmax activation function for the last layer guarantees that each neuron’s output activation can be interpreted as the probability of a particular input image belonging to that class.

2.2 Training

In order to train the classifier, we use all available slices of the training stack, i.e. 30 images with a 512×512 resolution. For each slice, we use all *membrane* pixels as positive examples (on average, about 50000), and the same amount of pixels randomly sampled (without repetitions) among all *non-membrane* pixels. This amounts to 3 million training examples in total, in which both classes are equally represented. Because the appearance of membranes is not affected by their direction, before each training epoch we further augment the training set by randomly mirroring each instance, and/or rotating it by $\pm 90^\circ$.

2.3 Postprocessing of network outputs

Because each class is equally represented in the training set but not in the testing data, the network outputs cannot be directly interpreted as probability values; instead, they tend to severely overestimate the membrane probability. To fix this issue, a polynomial function post-processor is applied to the network outputs. Its coefficients are computed as follows. A separate network is trained on a subset

of the training slices and tested in the remaining ones (for which ground truth is available). Resulting outputs (a total of 2.6 million instances) are compared to ground truth, to compute the transformation relating the network output value to the actual probability of being a membrane. The resulting function is well approximated by a monotone cubic polynomial (inset of Figure 1), whose coefficients are computed by least-squares fitting. The same function is then used to calibrate the outputs of all trained networks.

After calibration (a grayscale transformation in image processing terms), network outputs are spatially smoothed by a 2-pixel-radius median filter.

2.4 Foveation and nonuniform sampling

We experimented with two related techniques for improving the network performance by manipulating its input data, namely *foveation* and *nonuniform sampling* (see Figure 2).

Foveation is inspired by the structure of human photoreceptor topography [8], and has recently been shown to be very effective for improving nonlocal-means denoising algorithms [9]. It imposes a spatially-variant blur on the input window pixels, such that full detail is kept in the central section (*fovea*), while the peripheral parts are defocused by means of a convolution with a disk kernel, to remove fine details. The network, whose task is to classify the center pixel of the window, is then forced to disregard such peripheral fine details, which are most likely irrelevant, while still retaining the general structure of the window (context).

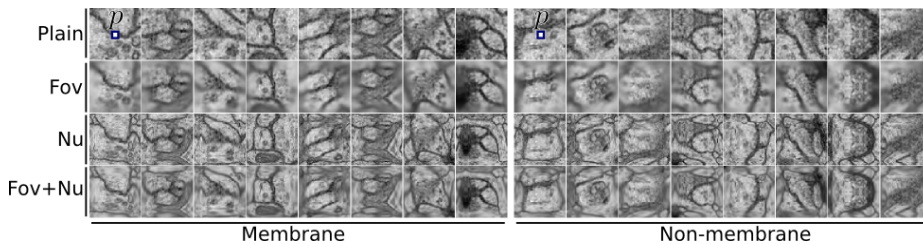


Fig. 2: Input windows with $w = 65$, from the training set. First row shows the original window (*Plain*); other rows show effects of foveation (*Fov*), nonuniform sampling (*Nu*), and both (*Fov+Nu*). Samples on the left and right correspond to instances of class *Membrane* and *Non-membrane*, respectively.

Nonuniform sampling is motivated by the observation that (in this and other applications) larger window sizes w generally result in significant performance improvements. However, a large w results in much bigger networks, which take longer to train and, at least in theory, require larger amounts of training data to retain their generalization ability. With nonuniform sampling, image pixels are directly mapped to neurons only in the central part of the window; elsewhere,

their source pixels are sampled with decreasing resolution as the distance from the window center increases. As a result, the image in the window is deformed in a fisheye-like fashion, and covers a larger area of the input image with fewer neurons.

Simultaneously applying both techniques is a way of exploiting data at multiple resolutions – fine at the center, coarse in the periphery of the window.

2.5 Averaging outputs of multiple networks

We observed that large networks with different architectures often exhibit significant output differences for many parts of the image, despite being trained on the same data. This suggests that our networks, being powerful and flexible classifiers, exhibit relatively large variance but low bias. It is therefore reasonable to attempt to reduce such variance by averaging the calibrated outputs of several networks with different structures.

This was experimentally verified. The submissions obtained by averaging the outputs of multiple large networks scored significantly better in all metrics than the single networks.

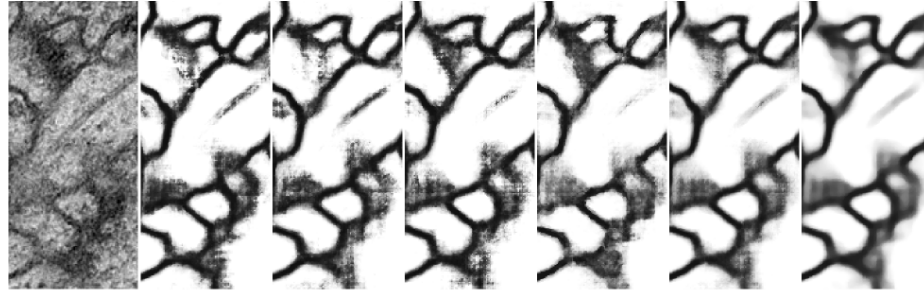
3 Experimental results

All experiments are performed on a computer with a Core i7 950 3.06GHz processor, 24GB of RAM, and four GTX 580 graphics cards. The GPU accelerates the forward propagation (FP) and BP routines by a factor of 50. Implementation details are described elsewhere [6].

We train four networks N1, N2, N3 and N4, with slightly different architectures, and window sizes $w = 65$ (for N1, N2, N3) and $w = 95$ (for N4); all networks use foveation and nonuniform sampling, except N3, which uses neither. As the input window size increases, the network depth also increases because we keep the convolutional filter sizes small. The architecture of N4 is the deepest, and is reported in Table 1.

Training time for one epoch varies from approximately 170 minutes for N1 ($w = 65$) to 340 minutes for N4 ($w = 95$). All nets are trained for 30 epochs, which leads to a total training time of several days. Classifying the 8 million pixels comprising the whole testing stack takes a relatively long time – 10 to 30 minutes on four GPUs – because the algorithm is not yet optimized to work on sliding windows. Dynamic programming can reduce this time by several orders of magnitude.

The outputs of four such networks are shown in Figure 3, along with their performance after filtering. By averaging the outputs of all networks, results improve significantly. The final result for one slice of the test stack is shown in Figure 4.



	Source	N1 $w=65$ Fov+Nu	N2 $w=65$ Fov+Nu	N3 $w=65$ Plain	N4 $w=95$ Fov+Nu	Averaged	Averaged +Filtered
Pixel err. $[\cdot 10^{-3}]$		65	66	66	68	62	} after filtering
Warping $[\cdot 10^{-6}]$		457	485	618	524	439	
Rand err. $[\cdot 10^{-3}]$		64	68	57	61	50	

Fig. 3: *Above*, from left to right: part of a source image from the test set; corresponding calibrated outputs of networks N1, N2, N3 and N4; average of such outputs; average after filtering. *Below*, the performance of each network, as well as the significantly better performance due to averaging their outputs. All results are computed after median filtering (see text).

Table 1: 11-layer architecture for network N4, $w = 95$.

Layer	Type	Maps and neurons	Kernel size
0	input	1 map of 95x95 neurons	
1	convolutional	48 maps of 92x92 neurons	4x4
2	max pooling	48 maps of 46x46 neurons	2x2
3	convolutional	48 maps of 42x42 neurons	5x5
4	max pooling	48 maps of 21x21 neurons	2x2
5	convolutional	48 maps of 18x18 neurons	4x4
6	max pooling	48 maps of 9x9 neurons	2x2
7	convolutional	48 maps of 6x6 neurons	4x4
8	max pooling	48 maps of 3x3 neurons	2x2
9	fully connected	200 neurons	1x1
10	fully connected	2 neurons	1x1

4 Discussion and conclusions

The main strength of our approach to neuron membrane segmentation in EM images lies in a deep and wide neural network trained by online back-propagation to become a very powerful pixel classifier with superhuman pixel-error rate, made possible by an optimized GPU implementation more than 50 times faster than equivalent code on standard microprocessors.

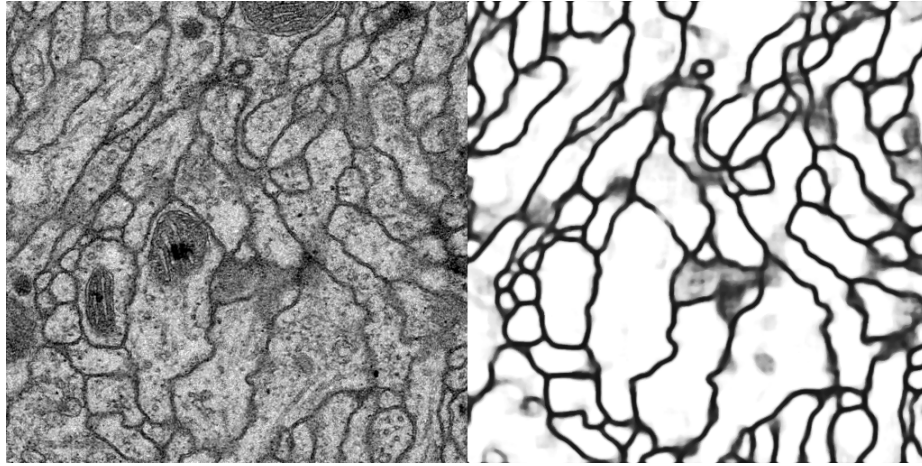


Fig. 4: Left: slice 16 of the test stack. Right: corresponding output.

In the competition, our approach outperformed all the others, despite not even being tailored to this particular segmentation task. Instead our DNN acts as a generic image classifier, using raw pixel intensities as inputs, without ad-hoc post-processing. This opens interesting perspectives on applying similar techniques to other biomedical image segmentation tasks. We intend to perform a proper analysis of the improvements due to foveation and nonuniform sampling.

Acknowledgments

This work was partially supported by a FP7-ICT-2009-6 EU Grant, Project Code 270247: A Neuro-dynamic Framework for Cognitive Robotics: Scene Representations, Behavioral Sequences, and Learning, and by a SNF grant: 200020-140399/1.

References

- [1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Ssstrunk. Slic superpixels. *Technical Report 149300 EPFL*, (June), 2010.
- [2] S. Behnke. *Hierarchical Neural Networks for Image Interpretation*, volume 2766 of *Lecture Notes in Computer Science*. Springer, 2003.
- [3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.
- [4] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.
- [5] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In *Proc. of*

- International Conference on Document Analysis and Recognition*, pages 1250–1254, 2011.
- [6] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proc. of International Joint Conference on Artificial Intelligence*, pages 1237–1242, 2011.
 - [7] D. C. Cireşan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, in press, 2012.
 - [8] C. Curcio, K. Sloan, R. Kalina, and A. Hendrickson. Human photoreceptor topography. *The Journal of comparative neurology*, 292(4):497–523, 1990.
 - [9] A. Foi and G. Boracchi. Foveated self-similarity in nonlocal image filtering. *Proceedings of SPIE*, volume 8291, 2012.
 - [10] K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.
 - [11] G. González, F. Fleurety, and P. Fua. Learning rotational features for filament detection. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 1582–1589, 2009.
 - [12] V. Kaynig, T. Fuchs, and J. Buhmann. Geometrical consistent 3d tracing of neuronal processes in sstem data. In *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 209–216, 2010.
 - [13] V. Kaynig, T. Fuchs, and J. Buhmann. Neuron geometry extraction by perceptual grouping in sstem images. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 2902–2909, 2010.
 - [14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
 - [15] A. Lucchi, K. Smith, R. Achanta, V. Lepetit, and P. Fua. A fully automated approach to segmentation of irregularly shaped cellular structures in em images. In *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 463–471, 2010.
 - [16] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.
 - [17] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proc. of International Conference on Artificial Neural Networks*, 2010.
 - [18] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2005.
 - [19] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proc. of International Conference on Document Analysis and Recognition*, pages 958–963, 2003.
 - [20] K. Smith, A. Carleton, and V. Lepetit. Fast ray features for learning irregular shapes. In *Proc. of International Conference on Computer Vision (ICCV)*, pages 397–404, 2009.
 - [21] D. Strigl, K. Kofler, and S. Podlipnig. Performance and scalability of GPU-based convolutional neural networks. In *Proc. of Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, 2010.