EFFICIENT CONVOLUTIONAL NEURAL NETWORKS FOR PIXELWISE CLASSIFICATION ON HETEROGENEOUS HARDWARE SYSTEMS

Fabian Tschopp[1] *Julien N. P. Martel*[2] *Srinivas C. Turaga*[3] *Matthew Cook*[2] *Jan Funke*[2,3]

[1]Department of Computer Science, ETH Zurich
[2]Institute of Neuroinformatics, University of Zurich and ETH Zurich
[3]Janelia Research Campus, Howard Hughes Medical Institute

ABSTRACT

With recent advances in high-throughput Electron Microscopy (EM) imaging it is now possible to image an entire nervous system of organisms like *Drosophila melanogaster*. One of the bottlenecks to reconstruct a connectome from these large volumes ($\approx$ 100 TiB) is the pixel-wise prediction of membranes. The time it would typically take to process such a volume using a convolutional neural network (CNN) with a sliding window approach is in the order of years on a current GPU. With sliding windows, however, a lot of redundant computations are carried out. In this paper, we present an extension to the Caffe library to increase throughput by predicting many pixels at once. On a sliding window network successfully used for membrane classification, we show that our method achieves a speedup of up to 57$\times$, maintaining identical prediction results.

Index Terms— convolutional neural networks, pixel wise classification, electron microscopy, loss functions

1. INTRODUCTION

Convolutional Neural Networks (CNN) are forward-backward neural networks that are based on convolutions using kernels, generally followed by element-wise non-linearities and pooling operations. The networks have been successfully employed for various image classification tasks [1]. Recent networks such as the U-Net [2] can feature over 20 layers, and contain millions of parameters to learn.

This paper focuses on the pixel-wise classification of images. For pixelwise classification, as opposed to image classification, a label for each pixel in a given image has to be predicted. Since the fields of view for the predictions are overlapping between neighboring pixels, optimizations are possible reducing redundant computations. This is particularly useful for predictions in electron microscopy (EM) datasets of neural tissue, which can easily reach 100 TiB, even for small organisms like *Drosophila melanogaster*. Pixel-wise predictions in these datasets is the current bottleneck in the automatic segmentation of neurons. With our implementation of *strided kernels* (SK), we are able to reduce the needed processing time significantly compared to currently used *sliding window* (SW) approaches. On a deep CNN that we designed for membrane predictions in EM images, we show a reduction in processing time by a factor of 57, delivering exactly identical results. On a cluster with 100 current GPUs, this corresponds to a reduction from $\approx$ 23 years (SW network) to $\approx$ 145 days (SK network). Additionally, we show that a further reduction to $\approx$ 19 days is possible using our strided kernels in *fully convolutional* downsampling networks [2], delivering qualitatively similar results.

Strided kernels allow predicting many labels of an image patch at once, and thus avoid a lot of the redundancy carried out by sliding window approaches, as illustrated in Figure 1. *Fully convolutional* networks also predict large patches efficiently [3], but do not have an equivalent sliding window network.

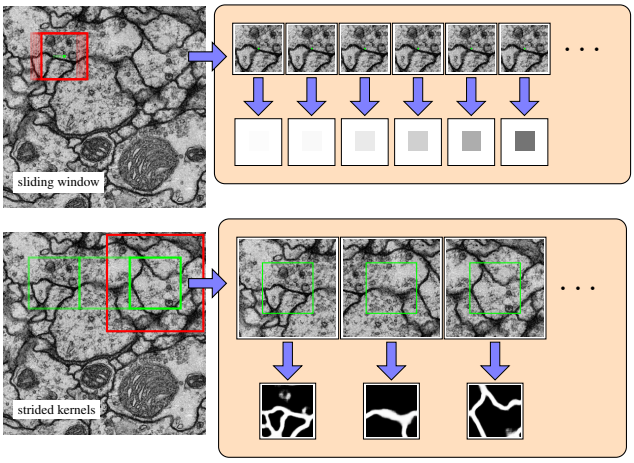

Fig. 1: The working principle of sliding window (SW) networks (top) and the proposed strided kernel (SK) networks (bottom). For a given input image, SW networks process a patch around every pixel (shown as red boxes). The fields of view of neighboring pixels have a considerable overlap where many computations are performed redundantly. In contrast, SK nets predict many pixels at once (green boxes), and thus decrease the number of redundant computations per pixel.

Thanks to AMD for hardware sponsoring of two W9100 GPUs.

Our approach is most similar to the strided kernels proposed by Hongsheng Li *et al*. [4]. Another similar method using a *fragment* dimension to store the shifted pooling outputs, instead of using a strided representation, has been proposed [5][6]. However, this method was implemented for CPUs and 2D only and is less suitable for GPUs in terms of memory access [4]. In contrast to the existing work, our implementation (1) supports convolutions and pooling in 3D, (2) is ready to be run on heterogeneous hardware (nVidia and AMD GPUs, CPU clusters), (3) features training on a topological loss by exploiting the pixel-wise predictions, (4) provides an easy-to-use tool targeted at users, not experts, to perform training and prediction, and (5) our source code is publically available.

Implementation wise, our work extends the open source neural network Caffe library, maintained by the Berkeley Vision and Learning Center [7]. We extended Caffe with our strided kernels for convolution and max pooling for three and more spatial dimensions, enabling our speedup also for classification in isotropic volumes. Support for different hardware backends is available through OpenCL and CUDA. Furthermore, we realize that the joint prediction of image patches can be exploited during training. To this end, we implemented the Malis criterion [8] using affinity graphs on the pixel-wise predictions as a new loss layer. This training loss is especially useful for biomedical image processing, where we want our classifiers to focus on minimizing topological errors during training, rather then pixel-wise errors. We refer to this extended library as Greentea.

2. METHOD FOR PIXEL-WISE CLASSIFICATION

We present our method by explaining the conversion of existing networks and demonstrate it on a 2D network that works well on our two datasets (see Section 4). In particular, we highlight how different types of layers (element-wise, convolution, pooling and inner product) change and what pixel-wise networks imply for training.

2.1. Example sliding window network design

We first designed a suitable sliding window network for neural tissue images (Figure 2), with a field of view the size of a mitochondria ($\approx 102^2$ pixels) in the datasets. This network is used as basis for benchmarks and to create an SK model, however this stands as an example and other networks can be used as a basis. The kernel sizes (in both dimensions) of our network are $k = 7, 5$ and 3 for the convolutions. For the max-pooling layers, we use a stride s of the same size as the kernel k to realize a downsampling by a factor of 2, i.e., $s = k = 2$.

2.2. Converting to strided kernel networks

Given a sliding window network with one pixel output and size $w_{\text{SW}}^{(0)}$ input, Algorithm 1 converts the network into a

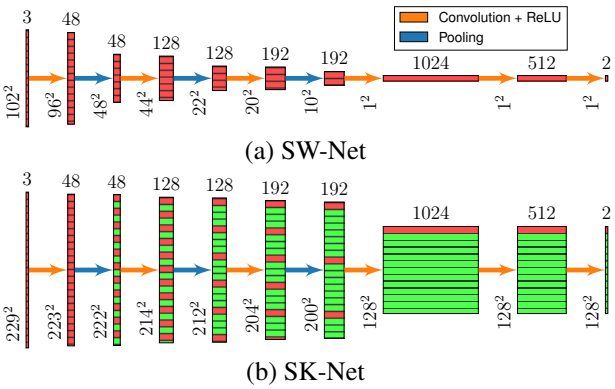

Fig. 2: Representation of (a) a sliding window network architecture and (b) a mathematically equivalent strided kernel network. The network contains 3 convolution-ReLU-pooling and 3 inner-product layers. Vertical numbers are the size of the feature maps, horizontal numbers the count of feature maps. Green-red striped blocks represent feature maps with a kernel stride ($d > 1$). The red blocks describe the path of a single pixel classification, as it would have been carried out by the SW-Net.

strided kernel network which is compatible with the existing trained weights and gives mathematically identical results to the original network, but processes images much faster for pixel wise classification. The number of feature maps and the field of view considered to predict a pixel remains identical.

In strided kernel networks, convolutions are performed on the whole input patch, and the results are shared between neighboring pixels. Max pooling (downsampling) is performed similarly, but different copies have to be generated because the pooling offsets are shifted for neighboring output pixels. For better memory access, the shifted copies are stored interleaved, which requires kernels with an inner stride d_{SK} [4] that keeps the data of each shifted copy separated.

The kernel stride consequently increases after each pooling operation (Algorithm 1, LN 15, Figure 2b) and has to be applied to all following layers. Inner product layers are replaced by convolutions with kernels equal to the feature map sizes in the original network (LN 18). The result is an independent fully connected layer for each of the shifted copies. Element-wise layers ($k = 1$) require no modification as they are not affected by the kernel stride.

The resulting network can theoretically be used with any output size $w_{\text{SK}}^{(N)}$. However, device memory required for convolution buffers and feature maps is often a limiting factor, as memory usage increases proportional to w^2. More memory allows a bigger patch size (w) to context padding (v) ratio, which avoids redundant computations.

On our example network, a $w^2 = 128^2$ output is feasible with 4 GiB GPU memory. The three inner product layers are replaced by convolutions with kernel sizes $k = 10, 1$ and 1 respectively, while the kernel stride grows up to $d = 8$ before collapsing to $d = 1$ after the first fully connected layer (Algorithm 1, LN 19).

Algorithm 1 Converts an SW-Net into a mathematically equivalent SK-Net by going through all layers $1, \ldots, N$ of the original network.

$\forall i \in [1, N].s_{\text{SK}}^{(i)} \leftarrow 1 \wedge p_{\text{SK}}^{(i)} \leftarrow 0$
$w_{\text{SK}}^{(0)} \leftarrow w_{\text{SW}}^{(0)}$ $d_{\text{temp}} \leftarrow 1$
for $i = 1; i \leq N; i \leftarrow i + 1$ **do**
  if $l_{\text{SW}}^{(i)} =$ convolution **then**
    $l_{\text{SK}}^{(i)} \leftarrow$ convolution SK
    $k_{\text{SK}}^{(i)} \leftarrow k_{\text{SW}}^{(i)}$ $d_{\text{SK}}^{(i)} \leftarrow d_{temp}$
    $w_{\text{SK}}^{(i)} \leftarrow w_{\text{SK}}^{(i-1)} - (k_{\text{SK}}^{(i)} - 1) \cdot d_{\text{SK}}^{(i)}$
    $\triangleright\ w_{\text{SW}}^{(i)} \leftarrow w_{\text{SW}}^{(i-1)} - (k_{\text{SW}}^{(i)} - 1)$
  else if $l_{\text{SW}}^{(i)}$ = pooling **then**
    if $w_{\text{SW}}^{(i-1)} \mod k_{\text{SW}}^{(i)} \neq 0 \vee k_{\text{SW}}^{(i)} \neq s_{\text{SW}}^{(i)}$ **then** *error*
    $l_{\text{SK}}^{(i)} \leftarrow$ pooling SK
    $k_{\text{SK}}^{(i)} \leftarrow k_{\text{SW}}^{(i)}$ $d_{\text{SK}}^{(i)} \leftarrow d_{temp}$
    $w_{\text{SK}}^{(i)} \leftarrow w_{\text{SK}}^{(i-1)} - (k_{\text{SK}}^{(i)} - 1) \cdot d_{\text{SK}}^{(i)}$
    $\triangleright\ w_{\text{SW}}^{(i)} \leftarrow \left\lceil \frac{w_{\text{SW}}^{(i-1)}}{k_{\text{SW}}^{(i)}} \right\rceil$
    $d_{temp} \leftarrow d_{temp} \cdot k_{\text{SK}}^{(i)}$
  else if $l_{\text{SW}}^{(i)}$ = inner product **then**
    $l_{\text{SK}}^{(i)} \leftarrow$ convolution SK
    $k_{\text{SK}}^{(i)} \leftarrow w_{\text{SW}}^{(i-1)}$ $\triangleright\ k_{\text{SW}}^{(i)} = w_{\text{SW}}^{(i-1)}$ is implicit
    $d_{\text{SK}}^{(i)} \leftarrow d_{temp}$ $d_{temp} \leftarrow 1$
    $w_{\text{SK}}^{(i)} \leftarrow w_{\text{SK}}^{(i-1)} - (k_{\text{SK}}^{(i)} - 1) \cdot d_{\text{SK}}^{(i)}$ $\triangleright\ w_{\text{SW}}^{(i)} \leftarrow 1$
  else
    if $k_{\text{SW}}^{(i)} > 1$ **then** *error*
    $l_{\text{SK}} \leftarrow l_{\text{SW}}$
    $w_{\text{SK}}^{(i)} \leftarrow w_{\text{SK}}^{(i-1)}$ $\triangleright\ w_{\text{SW}}^{(i)} \leftarrow w_{\text{SW}}^{(i-1)}$
if $d_{temp} = 1$ **then** *success* **else** *error*

2.3. Training strided kernel networks

In classical sliding window networks, the prediction of individual pixel labels is beneficial when training with a Softmax loss, because every pixel in the batch can be picked independently. When predicting complete tiles with strided kernel networks, however, pixels become spatially dependent and dictate the label distribution. This can lead to unbalanced loss backpropagation during training, as there are, for example, more cell interior pixels than membrane pixels. Learning rare features, such as synapses, is even more difficult. Our tool uses preprocessing methods to reduce these adverse effects.

For the training under a topological loss function, predictions of complete images are however required to determine if objects are properly separated. We implemented the Malis loss [8], as a training layer, taking advantage of our strided kernel networks.

3. GREENTEA BREW TOOL

To support training with pixel-wise labelled ground truth (instead of a single label per image), we provide an interface tool for Greentea. The tool can be configured for training, processing and benchmarking through prototxt files in the same style as Caffe networks. Our tool is designed to simplify pixel-wise training and prediction, making fast experimentation easy. The key features are:

1. **Easy setup:** The tool is easy to use, as it only requires a folder with training images and the corresponding label images. Tiles get cropped and processed through the network out of arbitrary sized input images (Figure 1).
2. **Pre-processing:** To be able to classify the whole image, border mirroring can be applied to extend the raw images. Image normalization and contrast limited adaptive histogram equalization (CLAHE) improve the training stability. The training data can furthermore be augmented through rotation, mirroring and blurring.
3. **Label preparation:** For balancing labels during training, the tool can prioritize image regions or mask error signals according to label frequencies in the training data to mimic independent and identically distributed training. Annotated labels can also be grouped together. The training data can be dense or sparse, by masking areas that are unlabeled or should be excluded from training.

4. RESULTS

4.1. Labeling throughput

Vendor	AMD	nVidia	Intel
Device	W9100 / R9 390X	GTX 980	i7-4790K
Compute Units	44 (2816)	16 (2048)	4 (8)
Memory (GiB)	16 / 8	4	16
FMA (GFLOP/s)	5240	4612	512
Memory (GiB/s)	320	224	25.6

Table 1: Hardware used for benchmarking.

The labeling throughput (Figure 3) is an overall performance measure for neural networks. It also shows how different devices and backends perform. Even on the CPU with OpenCL, using the strided kernel network gives a speedup of $35.75\times$ compared to sliding window networks.

CUDA performed best, but AMD OpenCL offers a competitive performance-price ratio, considering the customer-level R9 390X with the same performance as an AMD W9100. Performance is mainly determined by the BLAS matrix-matrix multiplications used to compute strided-kernel convolutions. For optimal performance, hardware specific BLAS libraries have to be used. Libraries such as cuDNN do not provide support for strided kernels yet.

4.2. Segmentation results

To assess the effect of using the Malis loss on complete image predictions, we evaluated the proposed SK network on two datasets: *ssTEM* [9] (*Drosophila melanogaster* larva ventral nerve cord, 20 images of 1024^2 pixels training data with binary segmentation) and *ISBI 2012* [10] (30 images of 512^2 pixels).

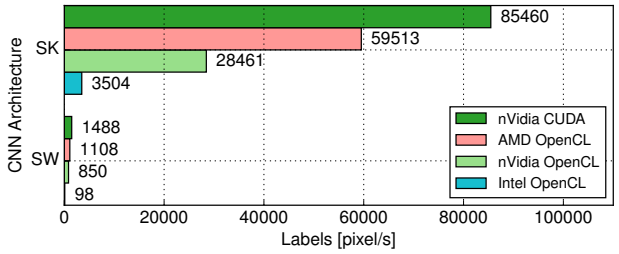

Fig. 3: Labeling throughput, using a batch of $n = 256$ with SW- and tiles of $(w + v - 1)^2 = (128 + 102 - 1)^2$ pixels with SK-Net.

Dataset	Loss Function	Rand	Warping	Pixel
ssTEM	Softmax + Malis	**0.043871**	**0.000324**	0.052724
ssTEM	Malis	0.063852	0.000386	0.055678
ssTEM	Softmax	0.123334	0.000549	**0.034416**
ISBI 2012	Softmax + Malis	**0.060110**	**0.000495**	0.106053
ISBI 2012	Malis	0.086975	0.000572	0.107365
ISBI 2012	Softmax	0.087380	0.000585	**0.075981**

Table 2: ssTEM & ISBI 2012 dataset error evaluation (sorted by Rand error, lower is better).

On both datasets (Table 2), training with Softmax and fine-tuning with Malis (Softmax + Malis) outperforms all other approaches assessed in our work considering the Rand error. The pixel accuracy decreases when switching from Softmax to Malis because Malis does not train pixel accuracy, but on topological correctness. Starting to train with Malis directly gives worse results as Malis requires some degree of topological separation on the prediction to produce stable error maps.

5. CONCLUSION

In this work we proposed to speed-up labeling throughput of existing sliding window networks in mathematically equivalent strided kernel networks. The accessibility of those deep neural networks has been greatly increased by supporting more hardware through OpenCL and providing an easy-to-use interface on top of the library. We considered the Malis criterion as an improved training method that works very well in combination with efficient pixel-wise networks due to the large tile outputs on which Malis can assess the topology during training. We demonstrated on two datasets that our networks exhibit a competitive accuracy. We show solid speedups of up to $57\times$ with strided kernel networks on heterogeneous hardware while maintaining identical numerical properties as sliding window networks. Although we presented the results in this paper in the context of EM images, the proposed techniques can be applied to any image processing problem that requires pixel-wise predictions.

6. SOURCE CODE

We provide our tools as an open source software stack:
`www.github.com/BVLC/caffe/tree/opencl`
`www.github.com/naibaf7/caffe`
`www.github.com/naibaf7/caffe_neural_tool`
`www.github.com/naibaf7/caffe_neural_models`
Full technical report:
`www.arxiv.org/abs/1509.03371`

7. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2015*, vol. 9351 of *Lecture Notes in Computer Science*, pp. 234–241. Springer International Publishing, 2015.

[3] Jonathan Long, Evan Shelhamer, and Trevor Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[4] H. Li, R. Zhao, and X. Wang, "Highly Efficient Forward and Backward Propagation of Convolutional Neural Networks for Pixelwise Classification," *ArXiv e-prints*, Dec. 2014.

[5] Alessandro Giusti, Dan Claudiu Ciresan, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber, "Fast image scanning with deep max-pooling convolutional neural networks," in *ICIP*, 2013, p. in press.

[6] Jonathan Masci, Alessandro Giusti, Dan Claudiu Ciresan, Gabriel Fricout, and Jürgen Schmidhuber, "A fast learning algorithm for image segmentation with max-pooling convolutional networks," in *ICIP*, 2013, p. in press.

[7] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, New York, NY, USA, 2014, MM '14, pp. 675–678, ACM.

[8] Srinivas C. Turaga, Kevin Briggman, Moritz N. Helmstaedter, Winfried Denk, and Sebastian Seung, "Maximin affinity learning of image segmentation," in *Advances in Neural Information Processing Systems 22*, pp. 1865–1873. Curran Associates, Inc., 2009.

[9] Stephan Gerhard, Jan Funke, Julien Martel, Albert Cardona, and Richard Fetter, "Segmented anisotropic sstem dataset of neural tissue," 2013.

[10] Albert Cardona, Stephan Saalfeld, Stephan Preibisch, Benjamin Schmid, Anchi Cheng, Jim Pulokas, Pavel Tomancak, and Volker Hartenstein, "An integrated micro- and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy," *PLoS Biology*, vol. 8, no. 10, pp. e1000502, oct 2010.