SegBlocks: Towards Block-Based Adaptive Resolution Networks for Fast Segmentation

Thomas Verelst(✉) and Tinne Tuytelaars(✉)

ESAT-PSI, KU Leuven, Leuven, Belgium
{thomas.verelst,tinne.tuytelaars}@esat.kuleuven.be

Abstract. We propose a method to reduce the computational cost and memory consumption of existing neural networks, by exploiting spatial redundancies in images. Our method dynamically splits the image into blocks and processes low-complexity regions at a lower resolution. Our novel BlockPad module, implemented in CUDA, replaces zero-padding in order to prevent the discontinuities at patch borders of which existing methods suffer, while keeping memory consumption under control. We demonstrate SegBlocks on Cityscapes semantic segmentation, where the number of floating point operations is reduced by 30% with only 0.2% loss in accuracy (mIoU), and an inference speedup of 50% is achieved with 0.7% decrease in mIoU.

1 Introduction and Related Work

Contemporary deep learning tasks use images with ever higher resolutions, e.g. 2048×1024 pixels for semantic segmentation on Cityscapes. Meanwhile, there is a growing interest for deployment on low-computation edge devices such as mobile phones. Typical neural networks are static: they apply the same operations on every image and every pixel. However, not every image region is equally important. Some dynamic execution methods [2,6,8] address this deficiency by skipping some image regions, but that makes them less suitable for dense pixel labelling tasks. Moreover, due to their implementation complexity, they do not demonstrate speed benefits [2], are only suitable for specific network architectures [8] or have low granularity due to large block sizes [6].

Some dynamic methods target segmentation tasks specifically. Patch Proposal Network [9] and Wu et al. [10] both refine predictions based on selected patches. Processing images in patches introduces a problem at patch borders, where features cannot propagate between patches when using zero-padding on each patch. Therefore, these methods use custom architectures with a global branch and local feature fusing to partly mitigate this issue. Huang et al. [3] propose to use a small segmentation network and further refine regions using a large and deeper network. However, since features of the first network are not re-used, the second network has to perform redundant feature extraction operations. Moreover, patches are 256×256 pixels or larger to minimize problems at patch borders which limits flexibility.

A. Bartoli and A. Fusiello (Eds.): ECCV 2020 Workshops, LNCS 12539, pp. 18–22, 2020.
https://doi.org/10.1007/978-3-030-68238-5_2

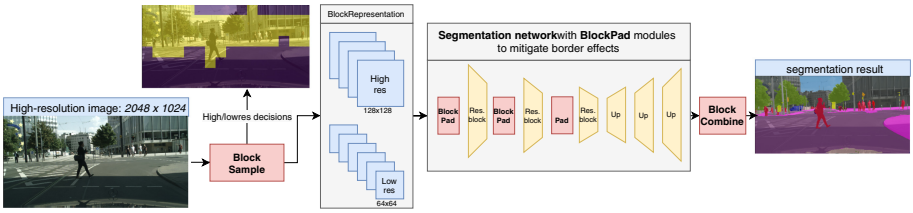

Fig. 1. Overview: The BlockSample module samples image regions in high or low resolution based on a complexity criterion. Image regions are efficiently stored in blocks. The BlockPad module enables feature propagation at block borders.

Our method differs from existing methods in two ways. First, there are no discontinuities at patch borders due to our novel BlockPad module, efficiently implemented in CUDA. BlockPad replaces zero-padding and propagates features as if the network was never processed in blocks, making it possible to adapt existing architectures without retraining. Also, we can obtain networks with different levels of complexity simply by changing a single threshold. Patches can be as small as 8×8 pixels, without major overhead. Secondly, instead of just skipping computations at low-complexity regions, we process a downsampled version of the region. This reduces the computational cost and memory consumption, while still providing dense pixel-wise labels. Our PyTorch implementation is available at https://github.com/thomasverelst/segblocks-segmentation-pytorch.

2 Method

Our method introduces several modules for block-based processing (Fig. 1):

BlockSample. The BlockSample module splits the image into blocks and low-complexity regions are downsampled by a factor 2 to reduce their computational cost. We use blocks of 128×128 pixels. As the network pools down feature maps, the block size is reduced accordingly up to just 4×4 pixels in the deepest layers, offering fine granularity. A simple heuristic determines the block complexity C_b. High-resolution processing is used when C_b exceeds a predefined threshold τ. For each image block b, we compare the $L1$ difference between the original image content of block I_b and a down- and re-upsampled version:

$$C_b = L_1(I_b - \texttt{nn\_upsampling}(\texttt{avg\_pooling}(I_b))) \tag{1}$$

BlockPad. The BlockPad module replaces zero-padding in order to avoid discontinuities at block borders, as shown in Fig. 2. For adjacent high-resolution blocks, BlockPad copies corresponding pixel values from neighboring blocks into the padding. When padding a low-resolution block, the two nearest pixels of the

neighboring block are averaged and the value is copied into the padding. Figure 3 demonstrates the benefit of BlockPad compared to standard zero-padding.

BlockCombine. This module upsamples and combines blocks into a single image.

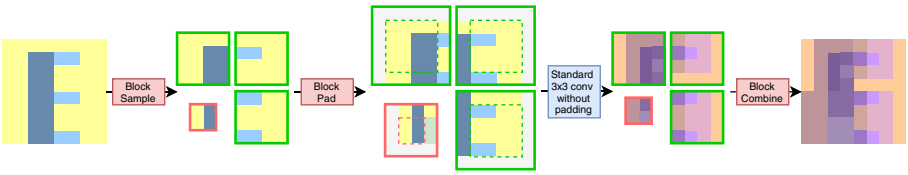

Fig. 2. Illustration of our modules. The BlockPad module replaces zero-padding for 3×3 convolutions and avoids discontinuities between patches.

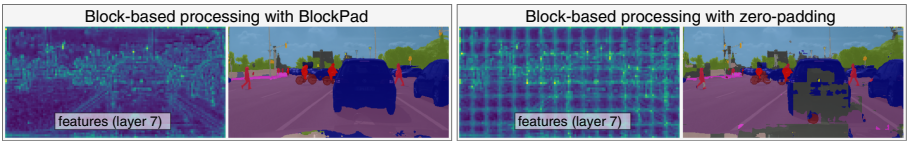

Fig. 3. Impact of the BlockPad module, by visualizing the intermediate feature maps and outputs. Zero-padding of individual blocks introduces border artifacts in the corresponding feature maps, resulting in poor segmentation results. The BlockPad module adds padding as if the image was never split in blocks.

3 Experiments and Results

We test our method on the Cityscapes dataset [1] for semantic segmentation, consisting of 2975 training and 500 validation images of 2048×1024 pixels. Our method is applied on the Swiftnet-RN18 [5] network. Table 1 shows that our SegBlocks models achieve state of the art results for real-time semantic segmentation, having 75.4% mIoU accuracy with 45 FPS and 61.4 GMACs (=GFLOPS/2) on a Nvidia 1080 Ti 11GB GPU. This outperforms other dynamic methods [3,4,9,10] as well as other static networks. We report FPS of other methods and their normalized equivalents based on a 1080 Ti GPU, similar to [5]. We did not use TensorRT optimizations. Memory usage is reported for batch size 1, as the total reserved memory by PyTorch. We also experiment with a larger model based on ResNet-50, where our method shows more relative improvement as 1×1 convolutions do not require padding. SegBlocks makes it possible to run the SwiftNet-RN50 model at real-time speeds of 21 FPS.

Table 1. Results on Cityscapes semantic segmentation

Method	Set	mIoU	GMAC	FPS	norm FPS	mem
SwiftNet-RN50 (our impl.)	val	77.5	206.0	14 @ 1080 Ti	14	2575 MB
SegBlocks-RN50 ($\tau = 0.2$)	val	77.3	146.6	17 @ 1080 Ti	17	1570 MB
SegBlocks-RN50 ($\tau = 0.3$)	val	76.8	121.7	21 @ 1080 Ti	21	1268 MB
SwiftNet-RN18 (our impl.)	val	76.3	104.1	38 @ 1080 Ti	38	2052 MB
SegBlocks-RN18 ($\tau = 0.3$)	val	75.5	61.5	45 @ 1080 Ti	45	1182 MB
SegBlocks-RN18 ($\tau = 0.5$)	val	74.1	46.2	60 @ 1080 Ti	60	1111 MB
Patch Proposal Network [9]	val	75.2	–	24 @ 1080 Ti	24	1137 MB
Huang et al. [3]	val	76.4	–	1.8 @ 1080 Ti	1.8	–
Wu et al. [10]	val	72.9	–	15 @ 980 Ti	33	–
Learning Downsampling [4]	val	65.0	34	–	–	–
SwiftNet-RN18 [5]	val	75.4	104.0	40 @ 1080 Ti	40	–
BiSeNet-RN18 [11]	val	74.8	–	66 @TitanXP	64	–
ERFNet [7]	test	69.7	27.7	11 @ Titan X	18	–

4 Conclusion

We proposed a method to adapt existing segmentation networks for adaptive resolution processing. Our method achieves state-of-the-art results on Cityscapes, and can be applied on other network architectures and tasks.

Acknowledgement. The work was funded by the HAPPY and CELSA-project.

References

1. Cordts, M., et al.: The cityscapes dataset for semantic urban scene understanding. In: IEEE CVPR 2016 Proceedings, pp. 3213–3223 (2016)
2. Figurnov, M., et al.: Spatially adaptive computation time for residual networks. In: IEEE CVPR 2017 Proceedings, pp. 1039–1048 (2017)
3. Huang, Y.H., Proesmans, M., Georgoulis, S., Van Gool, L.: Uncertainty based model selection for fast semantic segmentation. In: MVA 2019 Proceedings, pp. 1–6 (2019)
4. Marin, D., et al.: Efficient segmentation: learning downsampling near semantic boundaries. In: IEEE CVPR 2019 Proceedings, pp. 2131–2141 (2019)
5. Orsic, M., Kreso, I., Bevandic, P., Segvic, S.: In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. In: IEEE CVPR 2019 Proceedings, pp. 12607–12616 (2019)
6. Ren, M., Pokrovsky, A., Yang, B., Urtasun, R.: SBNet: sparse blocks network for fast inference. In: IEEE CVPR 2018 Proceedings, pp. 8711–8720 (2018)
7. Romera, E., Alvarez, J.M., Bergasa, L.M., Arroyo, R.: ERFNet: efficient residual factorized convnet for real-time semantic segmentation. IEEE Trans. Intell. Transp. Syst. **19**(1), 263–272 (2017)
8. Verelst, T., Tuytelaars, T.: Dynamic convolutions: exploiting spatial sparsity for faster inference. In: IEEE CVPR 2020 Proceedings, pp. 2320–2329 (2020)

9. Wu, T., Lei, Z., Lin, B., Li, C., Qu, Y., Xie, Y.: Patch proposal network for fast semantic segmentation of high-resolution images. In: AAAI 2020 Proceedings, pp. 12402–12409 (2020)
10. Wu, Z., Shen, C., Hengel, A.V.D.: Real-time semantic image segmentation via spatial sparsity. arXiv preprint arXiv:1712.00213 (2017)
11. Yu, C., Wang, J., Peng, C., Gao, C., Yu, G., Sang, N.: BiSeNet: bilateral segmentation network for real-time semantic segmentation. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11217, pp. 334–349. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01261-8_20