Content-Adaptive Downsampling in Convolutional Neural Networks

– Supplementary Material –

Robin Hesse[1] Simone Schaub-Meyer[1,2] Stefan Roth[1,2]

[1]Department of Computer Science, TU Darmstadt [2]hessian.AI
{robin.hesse, simone.schaub, stefan.roth}@visinf.tu-darmstadt.de

A. Overview

This appendix provides additional explanations and experimental details for reproducibility purposes, which could not be included in the main text due to space limitations.

B. Addendum to Extension to CNNs

In Sec. 3 of the main paper, we briefly discussed how our method can be applied multiple times in succession to extend it to CNNs. For a more detailed description and illustration of this process, please refer to Fig. 9.

C. Experimental Details

In the following, we provide additional details to ensure reproducibility. All experiments have been conducted using Python, PyTorch 1.11.0 [48], and a single NVIDIA RTX A6000 (48GB) GPU or a single NVIDIA A100-SXM4 (40GB) GPU.

Implementation of different output strides. In all experiments, we examine backbone models with varying output strides. Fig. 8 illustrates how these different output strides (OS) are realized. Fig. 8*(a)* shows a standard backbone using regular downsampling [17, 37] with the standard downsampling factor d=2. Part *(b)* shows how the backbone can be modified to retain a lower output stride, *i.e.*, higher resolution, by using dilated convolution [46]. Alternatively, it can be modified to contain multiple output strides using our novel adaptive downsampling scheme as shown in *(c)*.

C.1. Oracle experiment: Feature resolution in semantic segmentation

The experiments described in Sec. 4.1 of the main paper have been conducted on the widely used Cityscapes dataset [6] (freely available to academic and non-academic entities for non-commercial purposes), containing 5 000 annotated street scene images. Our code is built on top of the publicly available `DeepLabV3Plus-Pytorch` GitHub repository (MIT License).[1]

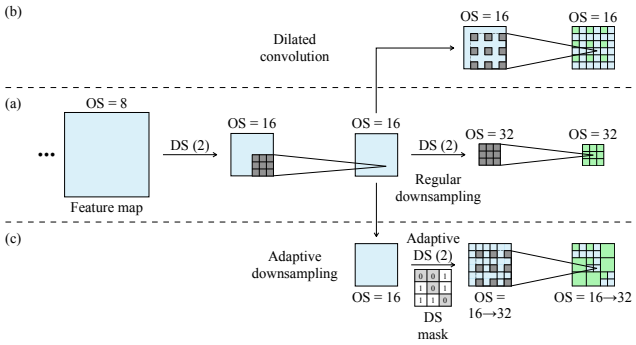

Figure 8. *Illustration of different methods to handle the output strides (OS) in CNNs. (a)* A standard convolutional network progressively downsamples the feature map. The illustrated convolutions, followed by uniform downsampling (DS), correspond conceptionally to strided convolutions with a stride of 2. *(b)* The last downsampling operation is substituted with a dilated convolution to retain the same receptive field while increasing the feature map resolution [46]. *(c)* The last downsampling operation is substituted with our novel adaptive downsampling and the dilation is increased to retain the same receptive field. Our approach retains high feature map resolution only where needed.

We examine different ResNet [17] backbones (50, 101, and 152 layers), three established segmentation models (FCN [26], DeepLabv3 [3], and DeepLabv3+ [4]), and two extensions, *i.e.* Softpool [38] and deformable convolution [7]. Following common practice, we initialize the backbones with weights obtained from pre-training on the ImageNet dataset [49]; they are publicly available in PyTorch [48]. We then fine-tune models with regular output stride on the official Cityscapes [6] training split for 30k iterations using a learning rate of 0.01 for the backbone, a learning rate of 0.1 for the classifier, a batch size of 8,

[1] https://github.com/VainF/DeepLabV3Plus-Pytorch

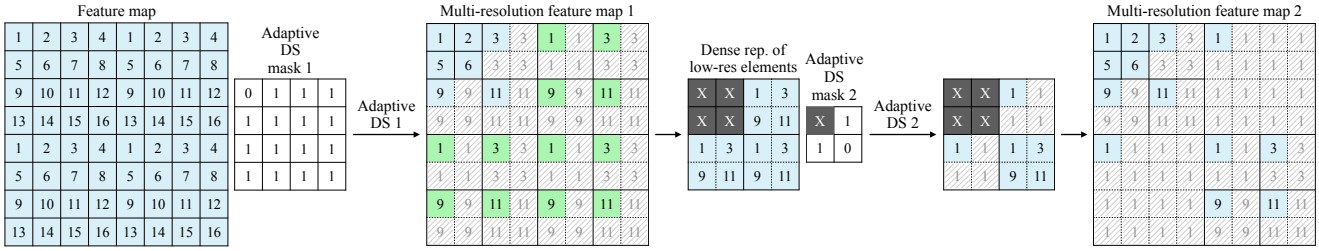

Figure 9. *Illustration of our adaptive downsampling applied two times in succession with a downsampling factor of* $d = 2$. Starting on the left, adaptive downsampling is applied a first time to a regular *feature map* to generate *multi-resolution feature map 1*. To perform adaptive downsampling a second ($k = 2$) time, we only consider the $d^k \times d^k$ patches of *multi-resolution feature map 1* that have previously been completely downsampled, *i.e.*, that are of lower effective resolution (green cells). The active elements of these patches are projected back to their regular, dense representation (*dense rep. of low-res elements*) to apply adaptive downsampling a second time ('X' denotes areas that cannot be represented at the current resolution as they have not been downsampled). The elements of the resulting feature map are then again projected onto their corresponding location to generate *multi-resolution feature map 2*.

and an SGD optimizer with a weight decay of $1e{-}4$ and a momentum of 0.9. For the final models with regular and adaptive downsampling, we fine-tune above models for another 10k iterations with learning rates multiplied by 0.1. The downsampling mask dilation sizes K for ResNet-{50, 101, 152} with DeepLabv3 are 33, 33, and 55, respectively. For ResNet-101 with FCN, respectively DeepLabv3+, K is chosen as 33. Finally, for ResNet-101 with DeepLabv3 and the extensions deformable convolution and SoftPool, we use $K = 55$. SoftPool is used as in the original paper [38]. Deformable convolution is only used in the last block of the ResNet backbone and offsets are estimated using a single convolutional layer with a kernel size of 3×3 and without a bias.

To measure the theoretical number of multiply-adds, we use the publicly available code in the `flops-counter.pytorch` GitHub repository (MIT License).[2] To estimate the runtime in seconds, we use 10 warm-up iterations, followed by another 10 iterations where we measure the average inference time of the backbone. For a fair comparison between equally optimized implementations, we use the same implementation for our proposed method and the baselines in the main paper. However, in practice, over the years there have been major efforts to highly optimize standard convolution in PyTorch [48]. To put our results into context with these optimizations, in Tab. 2 we report inference times for our method, and baseline methods with the default PyTorch implementations. As even the PyTorch version has major impact on the inference time of the baseline models, we report results for two versions, *i.e.*, 1.7.1 and 1.11.0.

Given a reasonable downsampling mask, our proposed method (OS=8→32) and implementation can outperform the default PyTorch baseline with an output stride of 8 (similar mIoU and 0.08s *vs*. 0.14s). For PyTorch 1.7.1, our implementation is even similarly efficient as the default implementation for the regular output strides (0.08s for OS=16 and 0.26 *vs*. 0.24 for OS=8). With version 1.11.0, PyTorch became more optimized, and thus, the gap to our implementation increases for the regular output strides (0.07s *vs*. 0.05s for OS=16 and 0.22s *vs*. 0.14s for OS=8). While we are not aware of the nature of the internal optimizations of PyTorch, it is reasonable to assume that similar optimizations could be applied to our implementation.

[2] https://github.com/sovrasov/flops-counter.pytorch

Table 2. *Inference times for different implementations.* We consider the ResNet-101+DeepLabv3 [3, 17] setup from the oracle experiment. We report mIoU, theoretical number of multiply-adds (#MA), and min./max. inference times for our and the default PyTorch [48] implementation with two different PyTorch versions.

OS	Impl.	PyTorch version	#MA	Time(s)	mIoU
16	*Ours*	1.7.1	4.2e11	0.08	0.7591
16	*Ours*	1.11.0	4.2e11	0.07	0.7591
16	*Default*	1.7.1	4.2e11	0.08	0.7591
16	*Default*	1.11.0	4.2e11	0.05	0.7591
8	*Ours*	1.7.1	1.4e12	0.26	0.7775
8	*Ours*	1.11.0	1.4e12	0.22	0.7775
8	*Default*	1.7.1	1.4e12	0.24	0.7775
8	*Default*	1.11.0	1.4e12	0.14	0.7775
8→32	*Ours*	1.7.1	6.7e11	0.09/0.18	0.7748
8→32	*Ours*	1.11.0	6.7e11	0.08/0.15	0.7748

C.2. Case study 1: Semantic segmentation

The experimental setup in Sec. 4.2 of the main paper follows the same setup as in Appendix C.1. All models are trained from scratch for 30k iterations. We investigate a ResNet-101 [17] backbone with the DeepLabv3 [3] segmentation model for OS=16 and OS=8. The runtime and theoretical number of multiply-adds are also measured as in Appendix C.1.

The downsampling masks for the edge detection setup are estimated by applying a Sobel filter to the original image, thresholding the result at 0.95, 0.35, respectively 0.15 to control the amount of invested resources, and dilating the result with a square dilation kernel of size 11×11.

To estimate the learned downsampling mask, we use a shallow CNN of 4 layers with a kernel size of 3×3, with 128, 64, 64, and 2 channels, with ReLU activation functions, and max pool with a stride of 2 after the first two layers. The output is fed into a Gumbel-Softmax [20] to obtain the final discretized downsampling mask. We train the mask estimator with a learning rate of 0.001 end-to-end together with the segmentation model. The two setups in Fig. 4 of the main paper are obtained by setting γ, *i.e.*, the fraction of active elements in the downsampling mask, to 0.5 and 0.7.

C.3. Case study 2: Keypoint description

For the keypoint description experiment in Sec. 4.3 of the main paper, we use the D2-Net descriptor [9]. Our code is based on the official, publicly available D2-Net [9] implementation (Clear BSD License).[3] We initialize our models with pre-trained weights that are also available in the official D2-Net [9] repository and do not train or fine-tune for any of the examined setups. We estimate the downsampling masks by dilating keypoints obtained from SIFT with filters of varying sizes for each downsampling level. To sample points with different computational complexity in Fig. 6 of the main paper, we use dilation sizes of – from least to most multiply-adds – (0,0,0), (0,0,5), (0,0,11), (0,0,21), (0,0,31), (0,19,35), (0,27,37), (19,41,41), (25,51,51), (∞,∞,∞). Here, the entries in each triplet correspond to the respective scale they operate on, *i.e.*, the first entry is the dilation size for the mask that retains elements at a resolution of OS=1, the second for retaining elements at a resolution of OS=2, and the third for retaining elements at a resolution of OS=4. An entry of 0 means that the entire feature map is downsampled while an entry of ∞ indicates that the resolution of the entire feature map is retained. As such, (0,0,0) corresponds to the regular output stride of 8, while (∞,∞,∞) corresponds to the regular output stride of 1. The used HPatches dataset [1] (MIT License), to the best of our knowledge, contains no personally identifiable information except for one sequence of an image of a prominent person that is already publicly available and not offensive.

[3] https://github.com/mihaidusmanu/d2-net

References

[48] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017. 1, 2

[49] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(13):211–252, 2015. 1