This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence

SegBlocks: Block-Based Dynamic Resolution Networks for Real-Time Segmentation

Thomas Verelst and Tinne Tuytelaars

Abstract—SegBlocks reduces the computational cost of existing neural networks, by dynamically adjusting the processing resolution of image regions based on their complexity. Our method splits an image into blocks and downsamples blocks of low complexity, reducing the number of operations and memory consumption. A lightweight policy network, selecting the complex regions, is trained using reinforcement learning. In addition, we introduce several modules implemented in CUDA to process images in blocks. Most important, our novel BlockPad module prevents the feature discontinuities at block borders of which existing methods suffer, while keeping memory consumption under control. Our experiments on Cityscapes, Camvid and Mapillary Vistas datasets for semantic segmentation show that dynamically processing images offers a better accuracy versus complexity trade-off compared to static baselines of similar complexity. For instance, our method reduces the number of floating-point operations of SwiftNet-RN18 by 60% and increases the inference speed by 50%, with only 0.3% decrease in mIoU accuracy on Cityscapes.

Index Terms—Conditional execution, convolutional neural networks, model compression, semantic segmentation

1 Introduction

Computational demands of computer vision are constantly increasing, with new deep learning architectures growing in size and new datasets containing images of ever higher resolution. For instance, the Cityscapes dataset for semantic segmentation includes images of 2048 by 1024 pixels [1], and high-resolution images are also used in medical [2] and remote sensing applications [3], [4]. The sheer number of pixels results in a large number of computations due to the convolutions. In addition, many convolutional layers are required to achieve a large receptive field [5].

Deep learning applications using these images are often deployed on low-power devices such as phones, surveillance cameras, or car platforms [6], [7]. The interest in embedded computer vision has lead to efficient neural network architectures [8], [9], [10], [11] and model compression methods [12], [13], [14].

These approaches result in static neural networks, which apply identical operations to every pixel. Yet not every image region is equally complex. Therefore, static networks may not be allocating operations in the most optimal way. Various methods have been proposed to dynamically adapt network architectures based on image complexity. Most are designed for classification tasks, for instance by completely skipping network layers or channels [15], [16], [17], [18], [19] and only few methods target dense pixel-wise classification tasks [20], [21], [22]. Dynamic methods typically focus on the theoretical reduction in computational complexity, due to the implementation challenges of sparse operations [18], [19], [23], [24]. In this work, we speed up inference using block-based processing, of which efficient implementations have already been demonstrated on GPU and other platforms [25], [26], [27].

• *T. Verelst and T. Tuytelaars are with the Center for Processing Speech and Images, Department Electrical Engineering, KU Leuven. E-mail: {thomas.verelst, tinne.tuytelaars}@esat.kuleuven.be*

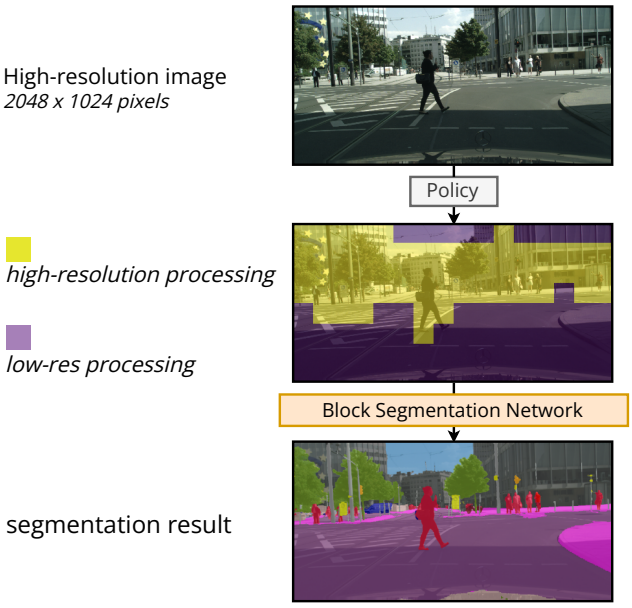
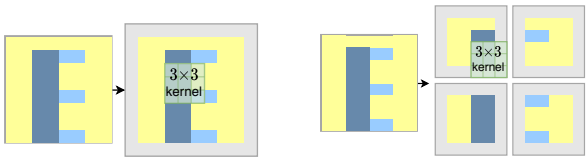


Fig. 1. SegBlocks adjusts the processing resolution of image regions based on their complexity. A lightweight policy network decides which blocks should be processed in high resolution mode. The number of operations is reduced without significant loss of accuracy.

We propose a method to dynamically process low-complexity regions at a lower resolution, as illustrated in Figure 1. Our method splits an image into blocks, and then downsamples non-important blocks. Reducing the processing resolution not only reduces the computational cost, but also decreases the memory consumption. The policy network, selecting the regions to be processed in high resolution, is trained using reinforcement learning.

Efficiently processing images in blocks without losing accuracy is not trivial. One could treat each block as a separate image and then combine the individual block outputs. How-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence



(a) Convolution on full image (b) Convolution on blocks

Fig. 2. Illustration of the zero-padding problem when processing blocks: convolutions pad individual blocks with zeros (grey), stopping the propagation of features between blocks and therefore resulting in a loss of global context.

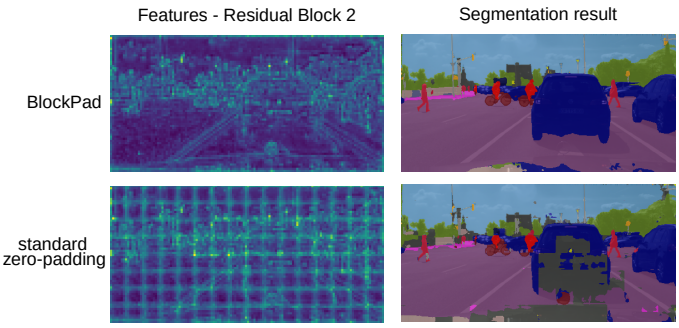


Fig. 3. Comparison of feature maps when processing an image in blocks with either standard zero-padding or our custom BlockPad module. Features are visualized by summing activation magnitudes over the channel dimension and tiling the outputs of individual blocks into a single image. Standard zero-padding introduces noticeable artifacts at block borders. In addition, the output shows inconsistencies, e.g. in the car, due to the lack of global context for individual blocks. In contrast, our BlockPad module enables feature propagation as if the network was never evaluated in blocks.

ever, features cannot propagate between individual blocks (Figure 2b), leading to a loss in global context and significant decrease in accuracy. Existing works partially address this by including global features extracted by a separate network branch, requiring custom architectures [21], [22].

Our method addresses this problem by replacing zero-padding with our BlockPad operation. This custom CUDA module copies features from neighboring blocks into the padding border and therefore preserves feature propagation between blocks, as if the network was never split into blocks. A comparison between zero-padding and the BlockPad module is given in Figure 3, showing that our module avoids artifacts at block borders.

The contributions of this work are as follows:

- We introduce the concept of dynamic block-based convolutional neural networks, where blocks are downsampled based on their complexity, to reduce their computational cost. In addition, we provide CUDA modules for PyTorch to efficiently implement block-based methods [1].
- We train the policy network with reinforcement learning, in order to select complex regions for high-resolution processing.
- We demonstrate our method using a state of the art semantic segmentation network, and show that our method reduces the number of floating-point operations (FLOPS) and increases the inference speed (FPS) with only a slight decrease in mIoU accuracy. Our method achieves better accuracy than static baseline networks of similar complexity.

This work is an extension of a 4-page short paper [28], where block-based processing was introduced in combination with a simple heuristic-based downsampling policy that selects regions where the loss of visual detail is most significant. Here, we provide more details on the block-based processing framework and we introduce a superior reinforcement learning downsampling policy that is trained specifically for the task at hand. In addition, we provide results for Cityscapes, CamVid and Mapillary Vistas datasets with more comprehensive ablation studies.

2 Related Work

2.1 Semantic segmentation

Traditional works in semantic segmentation focus on improving segmentation accuracy, without taking into account the network complexity [5], [29], [30]. In contrast to other tasks such as classification or object detection, segmentation requires pixel-wise labels of the same resolution as the input. Preserving spatial information in the network requires high-resolution latent representations with a high computational and memory cost. At the same time, a large receptive field is needed to incorporate global context. In order to keep the network size under control, semantic segmentation networks use encoder-decoder architectures with skip connections [30], dilated convolutions [31] and spatial pyramid pooling [5].

Applications such as driverless cars have raised interest in real-time inference on embedded devices. Several high-resolution datasets for these applications have emerged. The Cityscapes dataset [1] provides images of 2048×1024 pixels, with highly detailed annotations for 30 semantic classes. A more extensive dataset is Mapillary Vistas [32] with 25000 high-resolution images and 152 object categories. As real-time inference is crucial for these applications and datasets, smaller and more efficient segmentation networks have been developed for this task specifically.

ICNet [10] proposes a custom encoder architecture processing an image pyramid, with multi-resolution feature maps fused before the decoder. ERFNet [11] factorizes convolution kernels into 1×3 and 3×1 kernels to reduce the computational cost. Bilateral Segmentation Network (BiSeNet) [33] presents a network with two branches: a Spatial Path to encode high-resolution spatial information and a Context Path to achieve a high receptive field. A similar dual-branch architecture is proposed by Guided Upsampling Network (GUN) [34], where both branches share weights. A multi-branch network at different scales is demonstrated by ShelfNet [35]. DABNet [36] proposes a Depthwise Asymmetric Bottleneck module, which combines dilated convolutions with non-dilated ones, to capture local and more contextual information. ESPNetv2 [37] introduces a building block with dilated depthwise convolutions for large receptive fields. Other methods use attention modules [38], [39], [40], [41] to incorporate global context, for instance by weighting spatial features [42].

Many segmentation networks use classification back-

1. The code is available at https://github.com/thomasverelst/segblocks-segmentation-pytorch

bones. SwiftNet [43] demonstrates that state of the art results can be achieved with a straightforward architecture: a ResNet-18 encoder [44], pretrained on ImageNet [45], is combined with a spatial pyramid pooling (SPP) module [46] and basic decoder. Lightweight backbones such as MobileNetV2 [8], ShuffleNetV2 [9] and EfficientNet [47] use depthwise convolutions to reduce the computational cost, leading to models such as SwiftNetMN-V2 [43]. Model compression techniques can further reduce the computational cost, by applying pruning [12], knowledge distillation [13], quantization [14], or factorization [48], [49].

2.2 Dynamic segmentation

These efficient architectures and compression techniques achieve impressive performance, but do not exploit spatial properties of the image as they process each pixel with the same operation. Some recent methods address this by processing the image dynamically in blocks or patches. It is worth noting that dynamic methods are often complementary to static acceleration methods, and both can be combined to further reduce the computational cost.

The method proposed by Huang *et al.* [20] combines a fast segmentation model with a large model. First, the image is processed by a fast network, such as ICNet [10]. Then, a selection criterion, based on the softmax certainty of the output, uses the rough predictions to decide which image regions should be re-evaluated by the large model (e.g. PSPNet [29]). Such a two-stage approach has two drawbacks. First, the large network processes the image in blocks and the discontinuities at block borders stop feature propagation. Therefore, the method uses large patches of at least 256×256 pixels to include some global context. Secondly, the feature maps and predictions of the small network are not re-used by the large network, making the method less efficient and only applicable when the accuracy and cost difference between the small and large network is substantial.

Wu *et al.* [22] propose a custom architecture where a high-resolution branch improves rough predictions of a low-resolution branch. Again, this method struggles with the feature propagation problem at patch borders and therefore uses large blocks of 256×256 pixels. In addition, they incorporate global features from the low-resolution network into the high-resolution branch to provide more global context. The Patch Proposal Network [21] method of Wu *et al.* has a similar architecture, with a low-resolution global branch and high-resolution refinement branch. The local refinement branch only operates on selected patches, based on a trained selection criterion where regions with below-average accuracy are refined. Features of both branches are fused before generating final predictions.

In contrast to these methods, our dynamic method does not require modifications to existing network architectures. We address the discontinuities at block borders without requiring additional global features in the patch-based processing. Furthermore, by being applicable on existing architectures, our method benefits from further advancements in network architectures.

Another approach to reduce spatial redundancy is to warp the input image to enlarge important regions [50]. The work of Marin *et al.* [51] uses non-uniform image sampling to focus on complex regions. However, as the internal representation of the network is still a regular-spaced pixel grid, the flexibility of non-uniform downsampling is limited, and the network typically focuses on a single region.

2.3 Conditional execution

Adapting the network architecture based on the input image is also known as *conditional execution* [52], [53], [54] or dynamic neural networks. For instance, some methods reduce the processing cost of simple images by skipping complete residual blocks [15], [16], [17] or by dynamically pruning feature channels [18]. However, these methods are intended for classification tasks, with a large variety in features between images of different classes. In segmentation, many images contain the same objects and features in different spatial arrangements.

Some dynamic methods rely on spatial properties to reduce the computational cost: they skip computations on low-complexity regions. Spatially Adaptive Computation Time [24] and cascade-based methods [23] halt the computation of features when features are 'good enough'. DynConv [19] demonstrates inference speed improvements on human pose estimation tasks, where large regions of the image can be completely ignored by the network. The methods of Xie *et al.* [55] and Figurnov *et al.* [56] dynamically interpolate features in low-complexity regions. Requiring sparse convolutions, these methods either demonstrate no inference speed improvements [24], only on depthwise convolutions [19], or only on CPU [24], [55], or are intended for specialized hardware [57]. Block-based approaches are considered to be more feasible to implement efficiently on most platforms [25], [26], [27]. In addition, methods completely skipping non-complex regions are less suitable for pixel-wise labeling tasks such as segmentation. Our method processes every image region, but reduces the cost of non-complex regions.

An important aspect of conditional execution is the policy which determines the layers, channels or regions to execute. Often, this policy cannot be trained by standard backpropagation due to its discrete nature. Recently, the Gumbel-Softmax [58], [59] trick gained popularity in dynamic methods [16], [18], [19]. In our case, it would require predictions for each region in both high and low-resolution, increasing the computational cost at training time. Instead, we show that the policy can be trained efficiently using reinforcement learning [60].

3 Method

SegBlocks splits an image into blocks and adjusts the processing resolution of each block based on its complexity. A lightweight policy network processes a low-resolution version of the image and outputs resolution decisions for each image region. For implementation simplicity, we restrict the resolution to two options, with either high- or low-resolution processing. Note that high- and low-resolution regions are processed by the same convolutional filters, which consequently perceive objects at different scales. This does not affect the performance negatively though, as segmentation images typically have large variations in object

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence

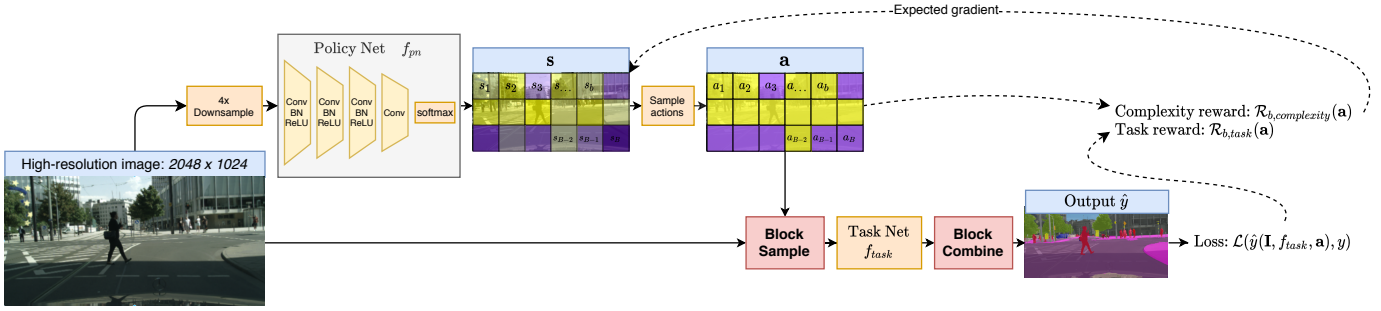


Fig. 4. Training the policy network with reinforcement learning: The lightweight policy net predicts soft decisions $\boldsymbol{s}$, which are sampled to actions $\boldsymbol{a}$. The policy net's gradients are estimated using REINFORCE [61] with a separate reward per block, based on the task loss in a block's region and the number of blocks processed at high resolution.

size, and convolutional neural networks are trained to be robust against scale variations. Since resolution decisions are discrete, the policy network is trained with reinforcement learning, using a reward function to determine the expected gradient.

First, we discuss how the reinforcement learning is incorporated using a reward taking into account both computational complexity and task accuracy. The policy is trained jointly with the main segmentation network. In the second part, we elaborate on custom modules making block-based adaptive processing of images possible.

3.1 Downsampling policy

3.1.1 Policy network

The policy network is a convolutional neural network that outputs a discrete decision per block. The network should be small compared to the main segmentation network. We use a simple architecture with 4 convolutional layers of 64 channels, batch normalization and ReLU non-linearities followed by a softmax layer over the channel dimension. A four times downsampled version of the image is given as input. The policy network f_{pn}, parametrized by θ, outputs probabilities s_b per block, indicating the likelihood that the block should be processed in high resolution based on the input image $\boldsymbol{I}$:

$$\boldsymbol{s} = f_{pn}(\boldsymbol{I}; \theta), \tag{1}$$

$$\text{with } \boldsymbol{s} = [s_1, \ldots, s_b, \ldots, s_B] \in [0,1]^B. \tag{2}$$

The soft probabilities are sampled to actions $\boldsymbol{a}$, where $a_b = 1$ results in high-resolution processing of block b:

$$s_b = P(a_b = 1). \tag{3}$$

Standard backpropagation cannot be applied due to the sampling of discrete actions. Finding optimal decisions $\boldsymbol{a}$ for context $\boldsymbol{I}$, with a constrained budget, is also referred to in literature as contextual bandits [60]. This can be seen as a reinforcement learning scenario with a single time step: the policy network predicts all actions a_b at once and directly obtains the reward. The probability of actions $\boldsymbol{a}$, for image $\boldsymbol{I}$ and parameters θ is given by

$$\pi_\theta(\boldsymbol{a} \mid \boldsymbol{I}) = \prod_{b=1}^{B} p_\theta(a_b \mid \boldsymbol{I}). \tag{4}$$

The objective is to find policy parameters θ, so that the predicted actions $\boldsymbol{a}$ for image $\boldsymbol{I}$ maximize the policy reward.

3.1.2 Policy reward

The goal of the policy network is to select the most important blocks for high-resolution processing. To avoid early convergence to a sub-optimal state where all blocks are processed at high resolution, the reward takes into account both the number of blocks processed at high resolution and the task accuracy. Figure 4 illustrates how both rewards are integrated in the training process.

We assume that a resolution decision a_b only affects the segmentation output of block b and does not influence adjacent blocks. The total reward for an image can then be expressed as the sum of individual block rewards. A block reward consists of one term optimizing accuracy and second term keeping the number of operations under control, weighted by hyperparameter γ (set to 10 in our experiments). The reward per image is then written as

$$\begin{aligned} \mathcal{R}(\boldsymbol{a}) &= \sum_{b}^{B} \mathcal{R}_b(\boldsymbol{a}) \\ &= \sum_{b}^{B} \big(\mathcal{R}_{b,task}(\boldsymbol{a}) + \gamma \mathcal{R}_{b,complexity}(\boldsymbol{a})\big). \end{aligned} \tag{5}$$

The percentage of blocks processed in high-resolution is given by

$$\sigma(\boldsymbol{a}) = \frac{1}{B} \sum_{b=1}^{B} a_b. \tag{6}$$

Then, the following reward minimizes the difference between σ and the desired percentage τ:

$$\mathcal{R}_{b,complexity}(\boldsymbol{a}) = \begin{cases} -(\sigma(\boldsymbol{a}) - \tau) & \text{if } a_b = 1, \\ \sigma(\boldsymbol{a}) - \tau & \text{if } a_b = 0. \end{cases} \tag{7}$$

This reward is positive for blocks processed at high resolution when the actual percentage σ is smaller than the desired percentage τ, resulting in an incentive for the policy network to process more blocks at high resolution. Using a target τ instead of simply minimizing σ results in more stable training and lower sensitivity to hyperparameter γ.

The task reward encourages the policy network to select regions with a high segmentation loss for high resolution processing. In general, those regions correspond to the most complex ones in the image. The task criterion, e.g. pixel-wise cross entropy, is denoted by $\mathcal{L}$. The task loss per image is then given by

$$L_{task} = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) \tag{8}$$

where $\widehat{\mathbf{y}}$ and $\mathbf{y}$ denote the predictions and ground truth labels respectively. Our reward is based on the task loss per block. Values $\widehat{\mathbf{y}}_b$ and $\mathbf{y}_b$ are obtained by only considering values in the block region. The task reward of block b can then be defined as

$$\mathcal{R}_{b,task}(\boldsymbol{a}) = \begin{cases} \mathcal{L}(\widehat{\mathbf{y}}_b, \mathbf{y}_b) - L_{task} & \text{if } a_b = 1, \\ -(\mathcal{L}(\widehat{\mathbf{y}}_b, \mathbf{y}_b) - L_{task}) & \text{if } a_b = 0. \end{cases} \quad (9)$$

Subtracting L_{task} is not strictly necessary but reduces the variance of the rewards for more stable training.

3.1.3 Expected gradient

The policy network should predict actions that maximize the expected reward:

$$\max \mathcal{J}(\theta) = \max \mathbb{E}_{\boldsymbol{a}\sim\pi_\theta}[\mathcal{R}(\boldsymbol{a})]. \quad (10)$$

The policy network's parameters θ can then be updated using gradient ascent with learning rate α:

$$\theta \leftarrow \theta + \alpha \nabla_\theta [\mathcal{J}(\theta)] \quad (11)$$

The gradients of J can be derived similarly to the policy gradient method REINFORCE [61]:

$$\begin{aligned} \nabla_\theta \mathcal{J}(\theta) &= \nabla_\theta \mathbb{E}_{\boldsymbol{a}}[\mathcal{R}(\boldsymbol{a})] \\ &= \nabla_\theta \sum_{\boldsymbol{a}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{I}) \mathcal{R}(\boldsymbol{a}) \\ &= \sum_{\boldsymbol{a}} \nabla_\theta \pi_\theta(\boldsymbol{a} \mid \boldsymbol{I}) \mathcal{R}(\boldsymbol{a}) \\ &= \sum_{\boldsymbol{a}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{I}) \nabla_\theta [\log \pi_\theta(\boldsymbol{a} \mid \boldsymbol{I}) \mathcal{R}(\boldsymbol{a})] \\ &= \sum_{\boldsymbol{a}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{I}) \nabla_\theta \Big[\log \prod_{b=1}^{B} p_\theta(a_b \mid \boldsymbol{I})\Big] \mathcal{R}(\boldsymbol{a}) \\ &= \sum_{\boldsymbol{a}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{I}) \mathcal{R}(\boldsymbol{a}) \sum_{b=1}^{B} \big(\nabla_\theta \log p_\theta(a_b \mid \boldsymbol{I})\big) \\ &= \mathbb{E}_{\boldsymbol{a}}\Big[\mathcal{R}(\boldsymbol{a}) \sum_{b=1}^{B} \big(\nabla_\theta \log p_\theta(a_b \mid \boldsymbol{I})\big)\Big] \\ &= \mathbb{E}_{\boldsymbol{a}}\Big[\sum_{b=1}^{B} \big(\mathcal{R}_b(\boldsymbol{a}) \nabla_\theta \log p_\theta(a_b \mid \boldsymbol{I})\big)\Big]. \end{aligned} \quad (12)$$

In practice, the expectation in Equation 12 is approximated with Monte-Carlo sampling using samples in the mini-batch. The result can be interpreted as applying REINFORCE on each block individually with reward $\mathcal{R}_b(\boldsymbol{a})$, giving unbiased but high-variance gradient estimates. In practice, we found the policy network stable to train for a large range of hyperparameters.

The segmentation network and policy network are jointly trained. The hybrid loss to be minimized is then given by

$$L_{hybrid} = L_{task} + \frac{\beta}{N} \sum_{n=1}^{N} \sum_{b=1}^{B} \big(-\mathcal{R}_b(\boldsymbol{a}) \log p_\theta(a_b \mid \boldsymbol{I})\big) \quad (13)$$

with N the batch size and β a hyperparameter weighting the loss terms (set to 4 in our experiments).

3.2 Block modules

We introduce three new modules for block-based processing with convolutional neural networks: BlockSample, BlockPad, and BlockCombine. Figure 5 gives an overview of a simple block-processing pipeline.

3.2.1 BlockSample

The BlockSample module splits the image into blocks and downsamples low-complexity blocks consecutively. Average pooling is used for efficient downsampling. We use a downsampling factor of 2, reducing the computational cost of low-complexity regions by a factor 4. The CUDA implementation fuses the splitting and downsampling steps into a single operation. Images are split into blocks of a predefined block size, for instance 128×128 pixels. Downsampling operations, such as max pooling, further reduce the spatial dimensions of blocks throughout the network.

3.2.2 BlockPad

The BlockPad module replaces zero-padding and eliminates discontinuities at block borders by copying features from neighboring blocks into the padding. Evaluating the network with only high-resolution blocks is equivalent to normal processing without blocks.

When two high-resolution or two low-resolution blocks are adjacent, pixels can be copied directly while preserving their spatial relationship. When copying features from low-resolution blocks into the padding of high-resolution blocks, features are nearest-neighbor upsampled in order to preserve the spatial relationship, as illustrated in Figure 5. When copying features from high-resolution blocks into the low-resolution block's padding, we considered two possibilities: strided subsampling and average sampling. Figure 6a illustrates that subsampling copies features in a strided pattern, whilst average sampling (Fig 6b) combines multiple pixels. Our experiments show that average sampling achieves better results with a small increase in overhead.

3.2.3 BlockCombine

The BlockCombine module upsamples low-resolution blocks to the same resolution as the high-resolution ones, and then combines all blocks into a single tensor. Our CUDA implementation merges the nearest neighbour upsampling and block combining steps to reduce the overhead of this operation.

4 Experiments

We integrate our dynamic processing method in SwiftNet [43], a state of the art network for real-time semantic segmentation of road scenes. We test semantic segmentation on the Cityscapes [1], Camvid [62], and Mapillary Vistas [32] datasets. In addition, we provide an ablation study, analyzing the overhead of block-based processing. Our method is implemented in PyTorch 1.5 and CUDA 10.2, without TensorRT optimizations. We report two model complexity metrics: the number of computations and the frames per second.

The number of computations is reported in billions of *multiply-accumulates (GMACs)* per image, being half the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence

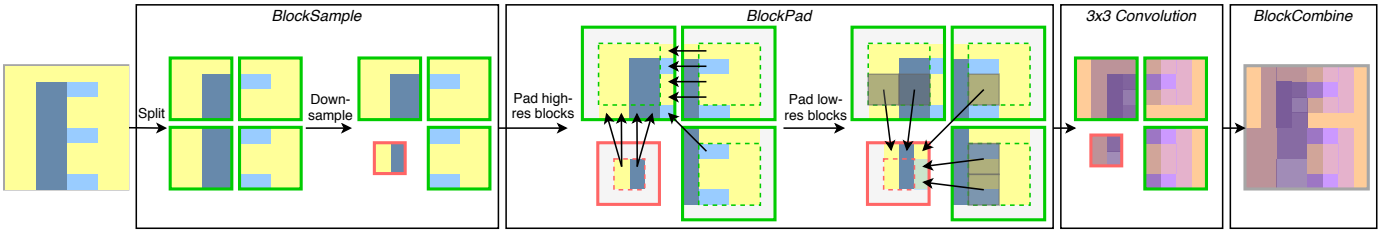
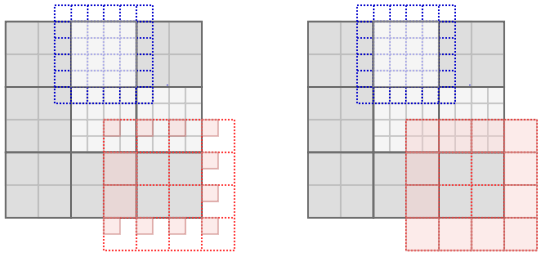


Fig. 5. Illustration of the BlockSample, BlockPad and BlockCombine modules. BlockSample splits the image and downsamples low-complexity blocks. BlockPad replaces zero-padding and enables feature propagation between blocks. The module copies features from neighboring blocks into the padding. When padding low-resolution blocks adjacent to high-resolution blocks, multiple pixel values of the high-resolution blocks are averaged to better preserve the spatial coherence of features.



(a) Strided subsampling (b) Average sampling

Fig. 6. BlockPad aims to respect the spatial relationship between high- and low-resolution blocks by sampling using the illustrated patterns. Low-resolution blocks (2×2 pixels size) are colored dark grey, while the 4×4 high-resolution blocks are colored light grey. The dotted blue and red grid shows the sampling pattern when padding high-resolution and low-resolution blocks respectively.

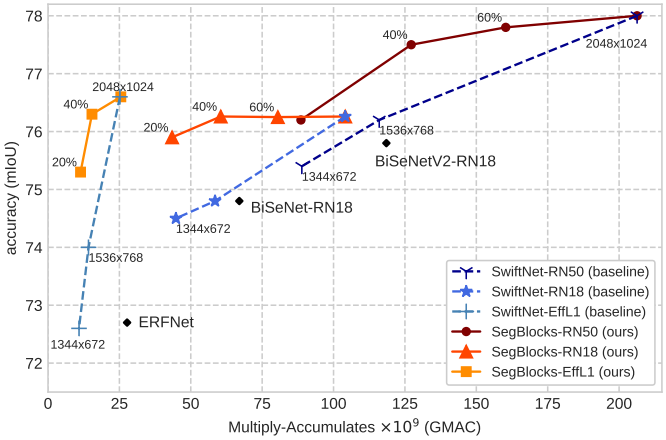


Fig. 7. Cityscapes validation results and comparison with static baselines of similar complexity. For a given computational complexity (GMACs), our adaptive resolution method SegBlocks consistently outperforms static baseline networks (SwiftNet) of similar complexity. Each backbone (RN50, RN18, Eff-L1) is trained and evaluated with $\tau \in 0.2, 0.4, 0.6$ for dynamic models and resolutions 2048×1024, 1536×768, 1344×672 for static baselines.

number of floating-point operations (FLOPS). The FLOPS metric is often used to compare model complexity in a platform-agnostic manner, since the efficiency of the implementation has no impact. For our dynamic method, where the number of computations varies per image, we report the average GMAC count, over the validation or test set.

Frames per second (FPS) is a more practical metric to measure inference speed, but depends largely on the implementation of building blocks. Therefore, both FPS and GMACs metrics should be taken into account for fair comparison. We measure the inference speed on both a high-end Nvidia GTX 1080 Ti 11GB GPU and low-end Nvidia GTX 1050 Ti 4 GB GPU, paired with an Intel i7 processor. The model is warmed up on the first half of the image set, and the speed is measured on the second half. To compare the inference speed to those reported by others, we normalize the FPS numbers (*norm FPS*) of different GPUs based on their relative performance, using the same scaling factors as Orsic *et al.* [43].

4.1 Cityscapes semantic segmentation

4.1.1 Dataset and setup

The Cityscapes [1] dataset for semantic segmentation consists of 2975 training, 500 validation and 1525 test images of 2048×1024 pixels. We use the standard 19 classes for semantic segmentation and do not use the additional coarsely labeled images.

We train SwiftNet [43] with three different backbones: ResNet-18 (RN18), ResNet-50 (RN50) [44], and EfficientNet-Lite1 (EffL1), which corresponds to EfficientNet-B1 [47] without squeeze-and-excite and swish modules. Our dynamic processing method is integrated in these baseline networks and we trained models for different $\tau \in [0, 1]$, indicating the desired percentage of high-resolution blocks. The models are trained on 768×768 crops, augmented by image scaling between factor 0.5 and 2, slight color jitter and random horizontal flip. The optimizer is Adam, the learning rate is cosine annealed from $4e{-}4$ to $1e{-}6$ over 350 epochs, weight decay is set to $1e{-}4$ and the batch size is 8. Standard cross entropy loss is used for models with a ResNet backbone, whereas the EfficientNet models use a bootstrapped cross entropy loss [65]. The backbone is pre-trained on ImageNet [45]. Our method uses a block size of 128×128 pixels, resulting in a block grid of 16×8 blocks per image that can adapt the processing resolution to the content.

4.1.2 Results and comparison

Figure 7 shows the mIoU accuracy of models at various computational costs (GMACs). Static baseline networks of different computational complexity are obtained by adjusting the training and evaluation resolution to 2048×1024, 1536×768 and 1344×672. The complexity of our SegBlocks methods is determined by the number of high-resolution

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 1
Results on Cityscapes semantic segmentation. Our SegBlocks models are based on the respective SwiftNet baselines, and integrate block-based dynamic resolution processing in those networks . The symbol '-' indicates that the metric was not reported. For our method, only several test set results are reported due to submission limitations on Cityscapes test evaluation.

	Method	mIoU *val*	*test*	GMACs	FPS	norm FPS
Our method	SwiftNet-RN50 (baseline, our impl.)	78.0		206.3	16.3 @ 1080 Ti, 3.8 @ 1050 Ti	16.3
	SegBlocks-RN50 (τ=0.4)	77.5	76.1	127.2	23.3 @ 1080 Ti, 5.7 @ 1050 Ti	23.3
	SegBlocks-RN50 (τ=0.2)	76.2		88.5	30.0 @ 1080 Ti, 7.3 @ 1050 Ti	30.0
	SwiftNet-RN18 (baseline, our impl.)	76.2	74.4	104.1	38.4 @ 1080 Ti, 9.9 @ 1050 Ti	38.4
	SegBlocks-RN18 (τ=0.4)	76.3	74.6	60.5	48.6 @ 1080 Ti, 13.5 @ 1050 Ti	48.6
	SegBlocks-RN18 (τ=0.2)	75.9		43.5	57.8 @ 1080 Ti, 16.8 @ 1050 Ti	57.8
	SwiftNet-EffL1 (baseline, our impl.)	76.6		24.9	17.4 @ 1080 Ti, 7.1 @ 1050 Ti	17.4
	SegBlocks-EffL1 (τ=0.4)	76.3	74.1	15.6	30.4 @ 1080 Ti, 8.4 @ 1050 Ti	30.4
	SegBlocks-EffL1 (τ=0.2)	75.3		11.4	35.6 @ 1080 Ti, 10.6 @ 1050 Ti	35.6
Dynamic networks	Patch Proposal Network (AAAI2020) [21]	75.2	-	-	24 @ 1080 Ti	24
	Huang et al. (MVA2019) [20]	76.4	-	-	1.8 @ 1080 Ti	1.8
	Wu et al. [22]	72.9	-	-	15 @ 980 Ti	33
	Learning Downsampling (ICCV2019) [51]	65.0	-	34	-	-
	HyperSeg-M [63]	76.2	75.8	7.5	36.9 @ 1080 Ti	36.9
	Stochastic Sampling [55]	80.6	-	373.2	no GPU implementation	-
Efficient static networks	ShelfNet18-lw (CVPR2019) [35]	-	74.8	95	36.9 @ 1080 Ti	36.9
	SFNet (ResNet-18) (ECCV2020) [35]	-	78.9	247	26 @ 1080 Ti	26
	SwiftNet-RN18 (CVPR2019) [43]	75.6	75.5	104.0	39.9 @ 1080 Ti	39.9
	ESPNetV2 (CVPR2019) [37]	66.4	66.2	2.7	-	-
	DABNet (BMVC2019) [36]	70.1	-	-	27.7 @ 1080 Ti	27.7
	BiSeNet-RN18 (ECCV2018) [33]	74.8	74.7	67	65.5 @ Titan Xp	58.5
	BiSeNetV2-RN18 (IJCV2021) [64]	75.8	75.3	118.5	47.3 @ 1080 Ti	47.3
	ICNet (ECCV2018) [10]	-	69.5	30	30.3 @ Titan X	49.7
	GUNet (BMVC2018) [34]	-	70.4	-	33.3 @ Titan Xp	29.7
	ERFNet (TITS2017) [11]	72.7	69.7	27.7	11.2 @ Titan X	18.4

TABLE 2
IoU per class on the Cityscapes validation set: our method improves the IoU score for most classes compared to lower-resolution baselines with similar complexity. Classes such as person, rider, car, bus and train benefit more from high-resolution processing. The percentage of pixels processed in high-resolution, given per class, shows that our method mostly processes road and sky regions at low resolution.

	road	side-walk	buil-ding	wall	fence	pole	traffic light	traffic sign	vege-tation	terrain	sky	per-son	rider	car	truck	bus	train	motor-cycle	bicycle	mIoU
SwiftNet-RN18 (2048×1024)	97.8	82.9	91.8	50.9	56.2	63.0	69.0	78.1	92.2	61.4	94.7	80.5	60.0	94.8	78.3	84.3	75.0	6.5	76.3	76.2
SwiftNet-RN18 (1536×768)	**97.8**	**83.0**	91.5	54.1	52.8	60.5	65.5	**76.4**	91.9	**62.1**	94.6	79.2	58.2	94.2	75.5	80.5	69.6	57.6	74.9	74.7
SegBlocks-RN18 (τ=0.4)	**97.8**	82.9	**91.9**	**59.5**	**57.6**	**61.1**	**67.0**	76.2	**92.1**	61.9	**94.4**	**80.4**	**59.9**	**94.4**	**77.2**	**82.4**	**77.2**	**60.1**	**75.7**	**76.3**
% high-res pixels	*8.5*	*45.8*	*64.0*	*67.0*	*88.5*	*86.3*	*85.6*	*82.2*	*59.7*	*68.2*	*36.7*	*93.3*	*96.2*	*66.2*	*61.7*	*68.9*	*85.0*	*90.9*	*93.2*	-

TABLE 3
Memory analysis for SegBlocks-RN18 running in 16-bit floating point, as reported by PyTorch. By storing some image areas at lower resolution, SegBlocks can reduce the memory usage, especially when using larger batch sizes.

model	Batch size 1	2	4
SwiftNet-RN18 (static)	480 MB	880 MB	1681 MB
SegBlocks-RN18 ($\theta = 0.4$)	571 MB	901 MB	1586 MB
SegBlocks-RN18 ($\theta = 0.2$)	340 MB	721 MB	940 MB

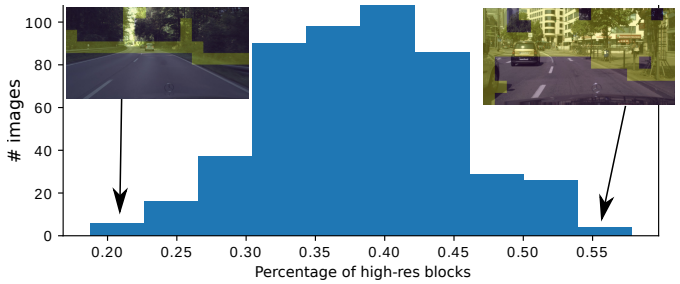


Fig. 8. Percentage of high-resolution blocks per image, as a histogram over 500 Cityscapes validation images, for SegBlocks-RN18 (τ=0.4). Simple images use fewer high-res blocks, down to 20%, and complex images have up to 55% high-res blocks (colored yellow).

blocks, changed by training for different values of hyperparameter $\tau \in \{0.2, 0.4, 0.6\}$. Our SegBlocks methods consistently outperform the static baseline networks, showing that dynamic resolution processing is more beneficial than sampling all regions at lower resolution. For instance, SegBlocks-RN18 with τ=0.4 reduces the computational complexity of the baseline network by 40%, without decreasing the mIoU. In contrast, a static baseline network of similar complexity, trained on images of 1536×768 pixels, achieves 1.7% lower mIoU.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence

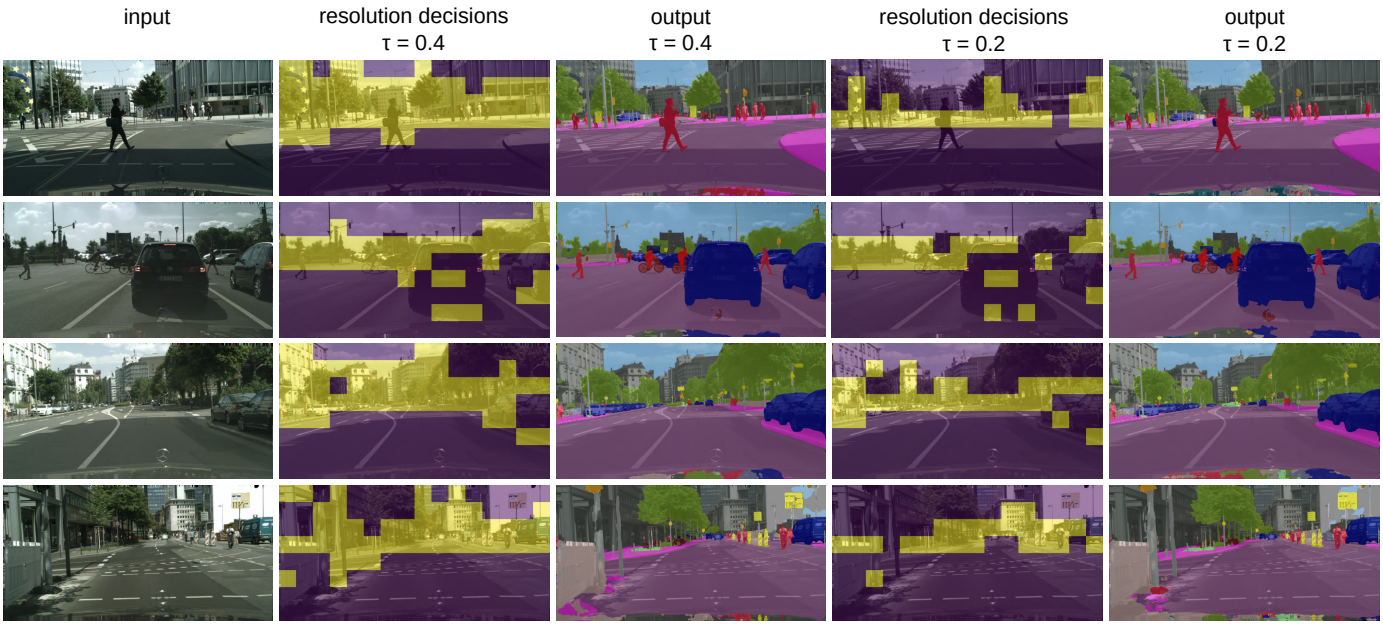


Fig. 9. Examples of resolution decisions made by the policy network and corresponding segmentation outputs, for SegBlocks-RN18 with $\tau = 0.4$ and $\tau = 0.2$. High-resolution blocks are colored in yellow.

Table 1 compares the performance of our method with other dynamic methods and non-dynamic efficient segmentation architectures. Compared to other dynamic methods, our method achieves higher mIoU results at faster inference speeds. Patch Proposal Network [21] is the most competitive method, and uses an architecture with a global branch and patch-based refinement branch. Our method achieves better accuracy and FPS due to our efficient block-based execution framework. This enables more fine-grained block-based execution, with in total 128 blocks (16×8 grid) compared to only 16 large patches, resulting in better adaptability and performance. Similarly, the method of Wu et al. [22] refines regions with a high-resolution branch on large patches. Huang et al. [20] use a cascade of two existing network architectures (e.g. lightweight ICNet and accurate PSPNet), where the second network is only applied on complex regions. However, this introduces additional latency and does not re-use any features of the lightweight network, resulting in low efficiency and only 1.8 FPS. The method of Marin et al. [51] (Learning Downsampling) adaptively samples the input image, to more densely sample near segmentation boundaries. Only the theoretical computational cost (GMACs) is reported. HyperSeg [63], based on the EfficientNet backbone [47], adjusts the weights of the decoder dynamically per image. Stochastic sampling [55] dynamically interpolates features spatially, but does not have an accelerated GPU implementation.

We are also competitive with state of the art architectures for fast semantic segmentation, such as BiSeNet [33] or ERFNet [11]. It is worthwhile to note that our proposed dynamic resolution method is complementary to further improvements in network architectures.

The table also demonstrates the inference speed improvement achieved using our efficient implementation. For instance, our method improves the inference speed of SwiftNet-RN50 from 16 FPS to 30 FPS on a GTX 1080 Ti, achieving real-time performance. This increase of 84 percent is obtained by reducing the theoretical complexity (GMACs) with 63%, while the mean IoU is decreased by 1.8%. Memory usage is reported in Table 3 as PyTorch's peak allocated memory during inference over the validation set, and indicates that our method also reduces memory consumption by storing low-complexity regions at lower resolution.

Our method adapts the operations to each individual image, and Fig. 8 shows the distribution of operations over images. Qualitative results are shown in Fig. 9, visualizing the resolution decisions of the policy network and the respective segmentation output, for τ set to 0.4 and 0.2. The policy network, trained with reinforcement learning, properly selects complex regions for high-resolution processing.

4.1.3 *Per-class analysis*

The IoU result per class is provided in Table 2, as well as the percentage of pixels processed in high-resolution for each class. Our method mainly processes classes such as road, sky and vegetation in low resolution, whereas rider, motorcycle and bicycle are typically sampled in high resolution.

4.2 CamVid semantic segmentation

Another popular benchmark for real-time semantic segmentation is CamVid [62], having 701 densely annotated frames, divided in 367 training, 101 validation and 233 test images. Following the methodology of related works [33], [43], [64], we train on the training and validation subsets and report test scores on 11 semantic classes. We evaluate on images of 960×704 pixels. We use the same hyperparameters and data augmentations as on Cityscapes, but with crops of 704 pixels. The backbone is pretrained on ImageNet, but we did not pretrain on Cityscapes. Dynamic methods use a block size of 64 pixels, resulting in a grid of 15×11 blocks. In order to adjust for the lower input resolution compared to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 4
Results on CamVid's test set for semantic segmentation [62]. All methods are pretrained on ImageNet [45], without Cityscapes pretraining [1]. Results annotated with † are determined using open-source implementations. FPS measured on Nivida GTX 1080 Ti.

Method	mIoU	GMACs	FPS
SwiftNet-EffL1 [43] (our impl.)	75.1	13.4	48
SegBlocks-EffL1 (τ = 0.4)	74.6 (-0.5%)	9.1 (-32%)	58 (+20%)
SegBlocks-EffL1 (τ = 0.2)	73.4 (-1.7%)	6.6 (-50%)	64 (+33%)
BiSeNet [33]	68.7	32.4†	116
BiSeNetV2 [64]	72.4		125
BiSeNetV2-L [64]	73.2		33
HyperSeg-S (ResNet-18) [63]	77.0		32.5
HyperSeg-S (EfficientNet-B1) [63]	78.4	6.3†	38.0
SFNet (DF2) [66]	70.4		134
SFNet (ResNet-18) [66]	73.8	41.2†	36

TABLE 5
Mapillary Vistas validation results. Results with † are reported in [67] and benchmark FPS with an Nvidia GTX2080 Ti GPU. Other results are benchmarked on an Nvidia GTX1080 Ti.

Network	Input size	FPS	GMACs	mIoU(%)
SwiftNet-EffL1 (our impl.)	1536×1152	20.7	21.5	41.7
SegBlocks-EffL1 (τ = 0.4)	1536×1152	33.0	13.7	40.5
SegBlocks-EffL1 (τ = 0.2)	1536×1152	40.1	11.5	39.8
AGLNet [68]	2048×1024	53.0	24.1G	30.7
WFDCNet [67]	2048×1024	53.1†	12.5G	30.5
FPENet [69]	2048×1024	92.5†	3.1G	28.3
DABNet [36]	2048×1024	78.3†	20.9G	29.6

Cityscapes, we remove the stride of the last residual block in the backbone, leading to higher-resolution representations in the spatial pyramid pooling module.

Table 4 shows that our results are competitive with other real-time semantic segmentation methods, and our dynamic execution (SegBlocks) improves inference speed over the static baseline network (SwiftNet). However, due to the smaller image dimensions and block size, the speedup is more modest compared to the high-res Cityscapes experiments.

4.3 Mapillary Vistas semantic segmentation

Mapillary Vistas [32] is a large dataset containing high-resolution road scene images with labels for semantic and instance segmentation. We use the standard 15000/2000 train/val split, resize all images to 1536×1152 pixels and use the 65 classes for semantic segmentation. We report validation results, since the test server is not available. SegBlocks usees blocks of 128 pixels, resulting in an adaptive grid of 12×9 blocks. The hyperparameters are the same as the Cityscapes experiments, with the batchsize reduced to 4 as the number of classes requires more memory, and we train SegBlocks with an EfficientNet-Lite1 backbone for 100 epochs. Table 5 demonstrates that SegBlocks reduces the number of operations and improves inference speed for the segmentation model on this dataset.

4.4 Ablation studies

Module execution time analysis: We profile the time characteristics of our block modules to analyze their overhead. Table 6 compares the execution time of the Policy Net, BlockSample, BlockPad and BlockCombine modules with the total runtime. The overhead of the Policy Net, BlockSample and BlockCombine modules is negligible, and the overhead of the BlockPad operation is reasonable with around 10 percent of the total execution time. The BlockPad module has less impact on the ResNet-50 backbone, as its 1×1 convolutions in the bottleneck function do not require padding. Note that the cost of the BlockPad operation scales with the total number of processed pixels, and thus with the number of high-resolution blocks.

TABLE 6
Time profiling of block modules, as total time taken during the inference of 250 validation images on an Nvidia GTX 1080 Ti GPU.

	SegBlocks-RN50		SegBlocks - RN18	
	τ=0.4	τ=0.2	τ=0.4	τ=0.2
mIoU	77.5%	76.2%	76.3%	75.9%
GMACs	127.2	88.6	60.5	43.5
policy GMACs	0.34	0.34	0.34	0.34
Runtime	11.6s	9.3s	5.8s	4.9s
Policy Net	0.04s (<1%)	0.04s (<1%)	0.04s (<1%)	0.04s (<1%)
BlockSample	0.12s (1%)	0.13s (1%)	0.11s (2%)	0.12s (2%)
BlockPad	0.75s (6%)	0.61s (7%)	0.68s (12%)	0.56s (11%)
BlockCombine	0.04s (<1%)	0.03s (<1%)	0.02s (<1%)	0.02s (<1%)

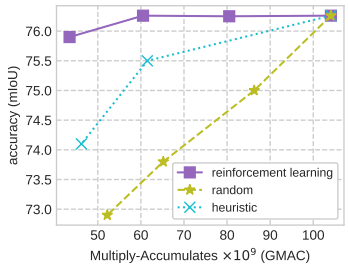


Fig. 10. Comparison of block execution policies. The reinforcement learning policy is proposed in this work. The random policy randomly selects a percentage of blocks per frame for high-resolution processing. The heuristic policy is given in [28].

Reinforcement learning policy: We compare the policy trained using reinforcement learning with other baselines in Fig. 10. The 'random' policy randomly selects blocks for high-resolution processing, with the amount of selected blocks given by the target percentage. The heuristic policy was proposed in [28] and selects the regions with the highest visual change, based on the average L2 distance between high- and low-resolution versions of a block.

Policy network architecture: The policy network, determining whether blocks should be executed with high- or low-resolution processing, should be lightweight but powerful enough for the selection task. The ablation (Table 7) shows that the size of this network does not significantly impact the results. For our experiments, we use an input resolution of 512x256 with a 4 layer network having 64 features in each layer.

Training steps and progress: We compare the training speed of a standard model with a dynamic SegBlocks model in Fig. 11, which shows that the reinforcement learning of the policy does not significantly impact the required amount of training steps.

Block size: Table 8 compares the impact of the method's block sizes on accuracy and performance. Fig-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3162528, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 7
Policy network ablation: adjusting the input resolution, number of layers and number of features for the policy network.

Policy input res.	# Layers	# Features	mIoU	Policy GMACs
128×64	2	64	75.3	0.01
256×128	3	64	75.5	0.07
512×256	4	64	76.3	0.34
1024×512	5	64	76.1	1.51
512×256	4	32	76.2	0.11
512×256	4	64	76.3	0.34
512×256	4	128	76.1	1.21

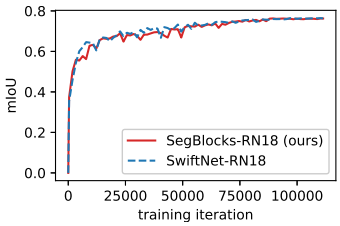


Fig. 11. Training progress given by validation mIoU for non-dynamic baseline (SwiftNet-RN18) and our dynamic method (SegBlocks-RN18). Learning the policy does not significantly impact the training progress.

TABLE 8
Block size comparison for SegBlocks-RN18 with τ=0.4 on the Cityscapes validation set .

Block size	mIoU	GMACs	FPS (1050 Ti)
64×64	76.3%	59.3	11.3
128×128	76.3%	60.5	13.5
256×256	75.2%	56.4	14.6

Fig. 12. Granularity of blocks of 64, 128, and 256 pixels on Cityscapes' images of 2048×1024 pixels.

ure 12 demonstrates the granularity of each block size on Cityscapes. Larger block sizes offer better inference speeds, by padding fewer pixels and having less overhead, whereas smaller block sizes offer finer control of adaptive processing.

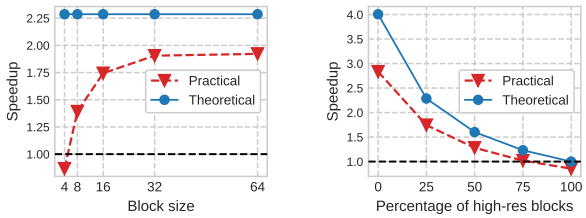
Sampling pattern: As described in Section 3.2.2, the BlockPad module pads individual blocks by sampling their neighbors. When low-resolution blocks are padded with features of high-resolution blocks, these features should be subsampled. Table 9 compares average-sampling with strided subsampling and shows that average-sampling achieves better accuracy (mIoU) with only a slight increase in overhead. In addition, we show that using zero-padding has a significant impact on accuracy, highlighting the importance of BlockPad when processing images in blocks.

4.4.1 *Single residual block analysis*

The overhead introduced by our block modules strongly depends on the block size: larger blocks have fewer pixels in the padding, and require fewer copy operations. Moreover, standard operations, implemented in PyTorch and cuDNN, are often optimized for larger spatial dimensions.

TABLE 9
Comparison of BlockPad sampling patterns and zero-padding for SegBlocks-RN18 (τ=0.4)

Method	mIoU	Runtime	BlockPad time
Avg-sampling	76.3%	5.8 s	0.68 s (12%)
Strided sampling	75.6%	5.7 s	0.63 s (11%)
Zero-padding	70.2%	5.2 s	N.A. (integrated in conv.)



(a) Block size impact (b) Percentage impact

Fig. 13. Comparison of theoretical versus practical speedup for a single residual block of ResNet-18, studying the impact of the block size and the percentage of high-resolution blocks on the performance.

We analyze a single residual block to measure the impact of both block size and the percentage of high-resolution blocks. Figure 13a studies the impact of the block size, when evaluating half of the blocks at high and half of the blocks at low resolution. The number of floating-point operations is reduced by 56%, resulting in a theoretical 2.3 times speed increase. In practice, we measure 1.92 times faster inference of the residual block when the block size is larger than 32. Block sizes smaller than 4 pixels result in slower inference. Note that, even though block sizes are typically large at the network input (e.g. 128×128) downsampling in the network reduces the block size by the same factor. The SwiftNet network downsamples by a factor 32 throughout the network, resulting in block sizes of 4×4 for the deepest network layers. Figure 13a shows that, when evaluating using block size 8, the percentage of high-res blocks should be lower than 75% to achieve practical speedup. For lower percentages, the practical speedup of our implementation is around 70 percent of the theoretical one.

5 Conclusion

We proposed a method to reduce the computational cost of existing convolutional neural networks, by evaluating images in blocks and adjusting the processing resolution of each block dynamically. We introduced custom block operations, enabling efficient inference and training of these dynamic neural networks. Our BlockPad module is essential to enable feature propagation between individual blocks. A lightweight policy network, trained with reinforcement learning, selects complex regions for high-resolution processing. Our method is demonstrated on semantic segmentation of street scenes and we show that the computational complexity of the SwiftNet architecture can be reduced with only a small decrease in accuracy. Dynamically adjusting the processing resolution per block achieves better accuracy than simply downscaling the image to a lower resolution.

The idea of block-based processing and dynamic resolution adaptation can be integrated in various network

architectures and computer vision tasks. In particular, our method is suited for dense pixel-wise classification tasks such as depth estimation, instance segmentation or human pose estimation. Furthermore, our block modules such as BlockPad pave the way towards other variants, for instance by processing blocks with varying quantization levels. Dynamic processing can help to keep the number of operations and energy consumption under control.

6 Acknowledgments

This work was funded by FWO on the SBO project with agreement S004418N.

References

[1] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 3213–3223.

[2] N. C. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler *et al.*, "Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on niomedical imaging (isbi)," in *15th Int. Symp. on Biomedical Imaging (ISBI 2018)*. IEEE, 2018, pp. 168–172.

[3] I. Demir, K. Koperski, D. Lindenbaum, G. Pang, J. Huang, S. Basu, F. Hughes, D. Tuia, and R. Raskar, "DeepGlobe 2018: A Challenge to Parse the Earth through Satellite Images," in *Proc. Conf. Computer Vision and Pattern Recognition Workshops (CVPRW)*. Salt Lake City, UT: IEEE, Jun. 2018, pp. 172–17 209.

[4] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark," in *Int. Geoscience and Remote Sensing Symp. (IGARSS)*. IEEE, Jul. 2017, pp. 3226–3229.

[5] L.-C. 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," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, Apr. 2018.

[6] A. Canziani, A. Paszke, and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," *arXiv:1605.07678 [cs]*, Apr. 2017.

[7] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT: IEEE, Jun. 2018, pp. 4510–4520.

[9] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proc. European Conf. on Computer Vision (ECCV)*, 2018, pp. 116–131.

[10] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for Real-Time Semantic Segmentation on High-Resolution Images," in *Proc. European Conf. Computer Vision (ECCV)*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer, 2018, vol. 11207, pp. 418–434.

[11] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation," *Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, Jan. 2018.

[12] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 598–605.

[13] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[14] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized Convolutional Neural Networks for Mobile Devices," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 4820–4828.

[15] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "SkipNet: Learning Dynamic Routing in Convolutional Networks," in *Proc. European Conf. on Computer Vision (ECCV)*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer, 2018, vol. 11217, pp. 420–436.

[16] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proc. European Conf. on Computer Vision (ECCV)*, 2018, pp. 3–18.

[17] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "BlockDrop: Dynamic Inference Paths in Residual Networks," in *Conf. Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT: IEEE, Jun. 2018, pp. 8817–8826.

[18] B. E. Bejnordi, T. Blankevoort, and M. Welling, "Batch-Shaping for Learning Conditional Channel Gated Networks," *arXiv:1907.06627 [cs, stat]*, Apr. 2020.

[19] T. Verelst and T. Tuytelaars, "Dynamic Convolutions: Exploiting Spatial Sparsity for Faster Inference," in *Conf. Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, Jun. 2020, pp. 2317–2326.

[20] Y.-H. Huang, M. Proesmans, S. Georgoulis, and L. Van Gool, "Uncertainty based model selection for fast semantic segmentation," in *16th Int. Conf. on Machine Vision Applications (MVA)*. Tokyo, Japan: IEEE, May 2019, pp. 1–6.

[21] T. Wu, Z. Lei, B. Lin, C. Li, Y. Qu, and Y. Xie, "Patch Proposal Network for Fast Semantic Segmentation of High-Resolution Images," *Proc. AAAI Conf. on Artificial Intelligence*, vol. 34, no. 07, pp. 12 402–12 409, Apr. 2020.

[22] Z. Wu, C. Shen, and A. v. d. Hengel, "Real-time Semantic Image Segmentation via Spatial Sparsity," *arXiv:1712.00213 [cs]*, 2017.

[23] X. Li, Z. Liu, P. Luo, C. C. Loy, and X. Tang, "Not All Pixels Are Equal: Difficulty-Aware Semantic Segmentation via Deep Layer Cascade," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 6459–6468.

[24] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially Adaptive Computation Time for Residual Networks," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 1790–1799.

[25] D. t. Vooturi, G. Varma, and K. Kothapalli, "Dynamic Block Sparse Reparameterization of Convolutional Neural Networks," in *Proc. Int. Conf. on Computer Vision Workshop (ICCVW)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 3046–3053.

[26] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "SBNet: Sparse Blocks Network for Fast Inference," in *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT: IEEE, Jun. 2018, pp. 8711–8720.

[27] F. Sun, M. Qin, T. Zhang, L. Liu, Y.-K. Chen, and Y. Xie, "Computation on sparse neural networks: an inspiration for future hardware," *arXiv preprint arXiv:2004.11946*, 2020.

[28] T. Verelst and T. Tuytelaars, "Segblocks: Towards block-based adaptive resolution networks for fast segmentation," in *Proc. European Conf. Computer Vision Workshop (ECCVW)*. Springer, Cham, 2020, pp. 18–22.

[29] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid Scene Parsing Network," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 6230–6239.

[30] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *Transactions Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[31] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[32] G. Neuhold, T. Ollmann, S. R. Bulo, and P. Kontschieder, "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes," in *Int. Conf. on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 5000–5009.

[33] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "BiSeNet: Bilateral Segmentation Network for Real-Time Semantic Segmentation," in *Proc. European Conf. on Computer Vision (ECCV)*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer Int. Publishing, 2018, vol. 11217, pp. 334–349.

[34] D. Mazzini, "Guided upsampling network for real-time semantic segmentation," *arXiv preprint arXiv:1807.07466*, 2018.

[35] J. Zhuang, J. Yang, L. Gu, and N. Dvornek, "Shelfnet for fast semantic segmentation," in *Proc. Int. Conf. on Computer Vision Workshops (ICCVW)*. IEEE, 2019.

[36] G. Li, I. Yun, J. Kim, and J. Kim, "Dabnet: Depth-wise asymmetric bottleneck for real-time semantic segmentation," *arXiv preprint arXiv:1907.11357*, 2019.

[37] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 9190–9200.

[38] Z. Huang, X. Wang, L. Huang, C. Huang, Y. Wei, and W. Liu, "Ccnet: Criss-cross attention for semantic segmentation," in *Proc. Int. Conf. on Computer Vision (ICCV)*. IEEE, 2019, pp. 603–612.

[39] F. Wu, F. Chen, X.-Y. Jing, C.-H. Hu, Q. Ge, and Y. Ji, "Dynamic attention network for semantic segmentation," *Neurocomputing*, vol. 384, pp. 182–191, Apr. 2020.

[40] Z. Zhu, M. Xu, S. Bai, T. Huang, and X. Bai, "Asymmetric Non-Local Neural Networks for Semantic Segmentation," in *Proc. Int. Conf. on Computer Vision (ICCV)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 593–602.

[41] P. Hu, F. Perazzi, F. C. Heilbron, O. Wang, Z. Lin, K. Saenko, and S. Sclaroff, "Real-time semantic segmentation with fast attention," *Robotics and Automation Letters*, vol. 6, no. 1, pp. 263–270, 2020.

[42] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local Neural Networks," *arXiv:1711.07971 [cs]*, Apr. 2018.

[43] M. Orsic, I. Kreso, P. Bevandic, and S. Segvic, "In Defense of Pre-Trained ImageNet Architectures for Real-Time Semantic Segmentation of Road-Driving Images," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, Jun. 2019, pp. 12 599–12 608.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016, pp. 770–778.

[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2009, pp. 248–255.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.

[47] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Int. Conf. on Machine Learning (ICML)*. PMLR, 2019, pp. 6105–6114.

[48] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets," in *Int. Conf. on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 6655–6659.

[49] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up Convolutional Neural Networks with Low Rank Expansions," *arXiv:1405.3866 [cs]*, May 2014.

[50] A. Recasens, P. Kellnhofer, S. Stent, W. Matusik, and A. Torralba, "Learning to zoom: a saliency-based sampling layer for neural networks," in *Proc. European Conf. on Computer Vision (ECCV)*, 2018, pp. 51–66.

[51] D. Marin, Z. He, P. Vajda, P. Chatterjee, S. Tsai, F. Yang, and Y. Boykov, "Efficient Segmentation: Learning Downsampling Near Semantic Boundaries," in *Int. Conf. on Computer Vision (ICCV)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 2131–2141.

[52] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional Computation in Neural Networks for faster models," *arXiv:1511.06297 [cs]*, Jan. 2016.

[53] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *arXiv:1308.3432 [cs]*, Aug. 2013.

[54] Y. Bengio, "Deep Learning of Representations: Looking Forward," *arXiv:1305.0445 [cs]*, Jun. 2013.

[55] Z. Xie, Z. Zhang, X. Zhu, G. Huang, and S. Lin, "Spatially adaptive inference with stochastic feature sampling and interpolation," in *Proc. European Conf. on Computer Vision (ECCV)*. Springer, 2020, pp. 531–548.

[56] M. Figurnov, A. Ibraimova, D. P. Vetrov, and P. Kohli, "Perforatedcnns: Acceleration through elimination of redundant convolutions," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 947–955.

[57] W. Hua, Y. Zhou, C. M. De Sa, Z. Zhang, and G. E. Suh, "Channel gating neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 1886–1896.

[58] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," *Proc. Int. Conf. Learning Representations (ICLR)*, 2017.

[59] C. J. Maddison, A. Mnih, and Y. W. Teh, "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables," *Proc. Int. Conf. Learning Representations*, 2017.

[60] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, ser. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 1998.

[61] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[62] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[63] Y. Nirkin, L. Wolf, and T. Hassner, "Hyperseg: Patch-wise hypernetwork for real-time semantic segmentation," in *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2021, pp. 4061–4070.

[64] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation," *Int. Journal of Computer Vision*, pp. 1–18, 2021.

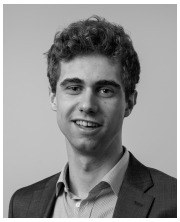
[65] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," *arXiv preprint arXiv:1412.6596*, 2014.

[66] X. Li, A. You, Z. Zhu, H. Zhao, M. Yang, K. Yang, S. Tan, and Y. Tong, "Semantic flow for fast and accurate scene parsing," in *Proc. European Conf. on Computer Vision (ECCV)*. Springer, 2020.

[67] X. Hao, X. Hao, Y. Zhang, Y. Li, and C. Wu, "Real-time semantic segmentation with weighted factorized-depthwise convolution," *Image and Vision Computing*, vol. 114, p. 104269, 2021.

[68] Q. Zhou, Y. Wang, Y. Fan, X. Wu, S. Zhang, B. Kang, and L. J. Latecki, "Aglnet: Towards real-time semantic segmentation of self-driving images via attention-guided lightweight network," *Applied Soft Computing*, vol. 96, p. 106682, 2020.

[69] M. Liu and H. Yin, "Feature pyramid encoding network for real-time semantic segmentation," *arXiv preprint arXiv:1909.08599*, 2019.



Thomas Verelst is a Ph.D. student at the Center for Processing Speech and Images (PSI) of KU Leuven university (Belgium). He received a M.Sc. degree in Electrical Engineering at the same university in 2018. His research focuses on efficient and dynamic network architectures for classification, pose estimation and segmentation tasks.

Tinne Tuytelaars is a full professor at the Electrotechnical department of KU Leuven. Her research focuses on image understanding, with a focus on representation learning, multimodal learning (images and text) and continual learning. In 2009, she received an ERC starting independent researcher grant, and in 2016, she received the Koenderink award. She has been one of the Program Chairs of the European Conference on Computer Vision 2014 and of IEEE/CVF Conference on Computer Vision and Pattern Recognition 2021, and one of the General Chairs of IEEE/CVF Conference on Computer Vision and Pattern Recognition 2016. She has been associate editor in chief of the IEEE Transactions on Pattern Analysis and Machine Intelligence and serves as area editor for the International Journal on Computer Vision.