ON THE COST OF HOMOGENEOUS NETWORK BUILDING BLOCKS AND PARAMETER SHARING

Anonymous authors

Paper under double-blind review

ABSTRACT

Deep neural networks have usually to be compressed and accelerated for their usage in low-power, e.g. mobile, devices. To this end, common requirements are low latency, high accuracy and a small memory footprint. A good trade-off between latency and accuracy has been shown with networks comprising multiple outputs at different depths. In this study, we further optimize such networks for efficient inference by introducing a homogeneous network structure to compute these outputs. A single building block is iteratively executed such that its output is refined over the iterations. On the `CamVid` and `Cityscapes` datasets for semantic segmentation, we show and compare empirical results for two scenarios: in the first scenario, the parameters are allowed to be updated at every iteration, whereas in the second scenario, parameters are kept constant and are shared between iterations. We compensate for both the homogeneity of the network architecture and the weight sharing by increasing the number of multiply-accumulate operations by a factor of 3, respectively. However, the potentially more efficient implementation of the introduced networks on novel, massively-parallel hardware accelerators may outweigh the increased number of operations.

1 INTRODUCTION

Deep neural networks with multiple outputs that, at the same time, heavily re-use intermediate activations are beneficial for a wide range of applications, especially in the mobile domain. In an online manner, they allow to trade off latency versus accuracy with barely any overhead (Huang et al., 2018; Ruiz & Verbeek, 2019). From an application point of view, the benefit of this trade-off can be best described in the following two scenarios (Huang et al., 2018): In the so-called *anytime prediction* scenario, the prediction of a network is progressively updated, whereas the first output defines the initial latency of the network. In a second scenario, a limited computational budget can be unevenly distributed over a set of samples with different "difficulties" in order to increase the average accuracy.

Instead of processing all scales in parallel (Ke et al., 2017; Huang et al., 2018; Ruiz & Verbeek, 2019) and increasing the depth of the network to obtain additional network outputs, Yu et al. (2019) keep the depth constant, but progressively increase the width of the network. However, for massively-parallel hardware systems (e.g. Esser et al., 2016), the wall-clock time for inference is likely to be dominated by the depth of the network. Ideally, if all paths from input to output of the network are computed in parallel, the longest path defines the latency from input to output (Fischer et al., 2018). Consequently, we focus on networks that compute all scales in parallel, which is also beneficial for the integration of global information, as especially required by dense prediction tasks like semantic segmentation (Zhao et al., 2017). In addition, the network building block between consecutive outputs is, ideally, also chosen to be as shallow as possible.

In this study, in addition to anytime prediction, we impose an additional restriction on neural networks to heavily re-use a single building block (see Figure 1). This means that the structure of computations, i.e. the type of network layers including their dimensions and connectivity, is identical for each re-use of this building block. However, we differentiate between the case, in which parameters are allowed to be updated at every iteration during training and inference, and the case, in which the parameters are kept constant and are shared between iterations. Conventional deep neural networks for image processing usually have barely any layers with identical properties, e.g. DenseNets (Huang et al.,

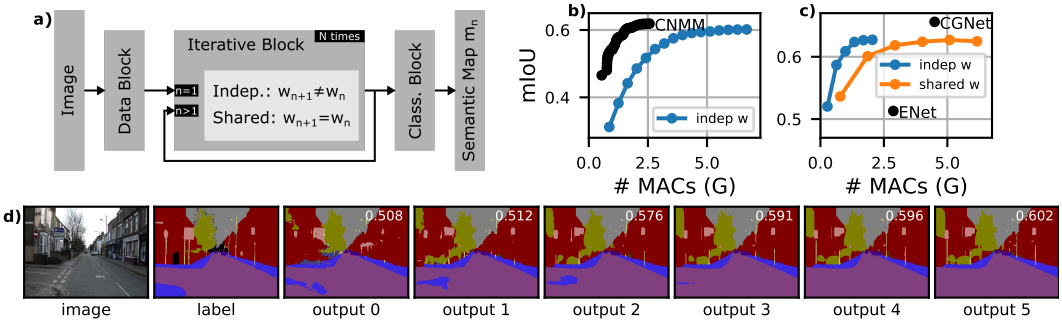

Figure 1: This study in a nutshell. **a)** The experimental setup: first, images are pre-processed and potentially down-scaled by a sub-network called *data block*. Then, the output of this data block is processed by another sub-network that is iteratively executed. After every iteration n, the output of this *iterative block* is fed back to its input and, at the same time, is further processed by the *classification block* to predict the semantic map m_n. This network generates multiple outputs m_n with increasing quality and computational costs and heavily re-uses intermediate network activations. We compare two options of this network: We either allow the weights of the iterative block to be independent between iterations or we share the weights across all iterations. Parameters of the classification block are never shared. **b)** Mean intersection-over-union (mIoU) over multiply-accumulate operations (MACs) on the validation set of the `Cityscapes` dataset. CNMMs (Ruiz & Verbeek, 2019) are to our knowledge the state-of-the-art baseline for efficient networks with multiple outputs and are approximately on par with the performance of the single-output ESPNetv2 (Mehta et al., 2019). Re-using an iterative building block (blue curve) requires approximately 3 times the number of MACs to recover to the baseline mIoU of CNMMs. **c)** Mean IoU over MACs on the test set of the `CamVid` dataset. ENet (Paszke et al., 2016) and CGNet (Wu et al., 2018) denote efficient reference networks that do not use pre-training on other datasets like MobileNetV3 (Howard et al., 2019). Weight sharing (orange curve) requires approximately 3 times the number of MACs to recover to the baseline mIoU of a network without weight sharing (blue curve). **d)** Network predictions over network outputs. The data sample with median mIoU of the network with independent weights in (c) is shown.

2017), or consist of many unique building blocks, e.g. one building block for each scale (Greff et al., 2017). In addition, parameters are usually not shared between repeated layers of identical structure, except for Liao & Poggio (2016).

We define networks to be *homogeneous* if they heavily re-use a unique building block. Although these building blocks are executed multiple times, their memory footprint is usually smaller than that of the layers of conventional networks. This is especially beneficial for massively-parallel hardware systems that usually have limited local memory, but provide extremely high throughput if the whole network fits into memory (Schemmel et al., 2010; Merolla et al., 2014; Yao et al., 2020). In terms of energy efficiency, the saving by the high throughput outweighs the additional costs of multiple iterations. Since analog hardware accelerators usually operate on shorter time scales than digital ones, e.g. compare Schemmel et al. (2010) and Yao et al. (2020) to Merolla et al. (2014), they are usually even more energy efficient. However, for analog systems, the update of weights between iterations of the building block is usually limited or not possible at all. In addition to an increased energy efficiency, a homogeneous network structure also decreases the complexity of mapping the network description to a hardware system, since, in principal, this has to be performed only once for each unique network building block.

In deep learning literature, the computational costs are usually quantified by counting the multiply-accumulate operations (MACs) required for the inference of a single sample and by counting the parameters. For fully convolutional networks, as commonly used for semantic segmentation tasks, the number of parameters are independent of the spatial resolution of the network's input as well as intermediate feature maps and, consequently, is not suited as a measure for computational costs. MACs have the advantage that they can be easily calculated and do not require any knowledge about the hardware system, on which the neural network is executed. However, for most hardware systems, not the MACs, but memory transfers are dominating the computational costs in terms of energy consumption (Chen et al., 2016; Chao et al., 2019). Unfortunately, memory transfers are

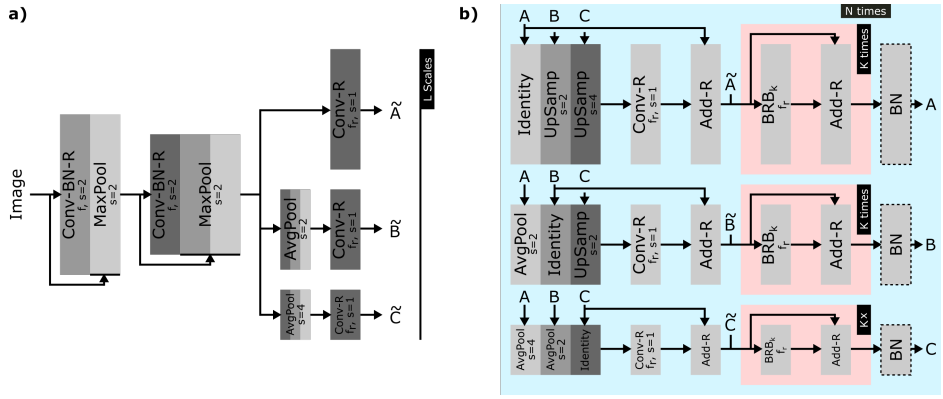

Figure 2: Detailed description of the *data block* (a) and *iterative block* (b) as shown in Figure 1: **a)** First, images are down-sampled in two stages, each using a stride and pooling of size $s = 2$ (inspired by Paszke et al., 2016). Then, the output of the second stage is processed on L different scales. For clarity, an example with $L = 3$ is shown. Convolutional layers (with kernel size 3), batch-normalization layers and ReLU activation functions are denoted with `Conv`, `BN` and `R`, respectively. **b)** The iterative block receives input at L different scales denoted as `A`, `B` and `C`. In order to mix the information between the different scales (e.g. Zhao et al., 2017) the inputs `A`, `B` and `C` are concatenated and processed with a residual block (He et al., 2016). Then, an additional K *bottleneck residual blocks* (BRB; with expansion factor $t = 6$) (Sandler et al., 2018) are applied to obtain the output of the iterative block consisting of feature maps at again L different scales. This output is both fed back as input for the next iteration n and forwarded to the classification block. The classification block consists of a 1×1 convolutional layer with C output channels and an 8-fold bilinear up-sampling to the original spatial dimensions. In the first iteration ($n = 1$) of the iterative block, the mixing of the scales is skipped and the input is directly processed by the K bottleneck residual blocks (see injection of input as $\tilde{\mathtt{A}}$, $\tilde{\mathtt{B}}$ and $\tilde{\mathtt{C}}$). The number of output channels are denoted by f and $f_{\mathrm{r}} = f/2$, respectively. The batch-normalization layers within the iterative blocks (dashed boxes) are never shared between iterations.

usually not reported and are sometimes impossible to reconstruct due to missing information about the network architectures. Consequently, we compare our networks on MAC-level and additionally provide analyses on the memory transfers.

In the following, we investigate the computational costs of highly re-using a single building block and of additionally sharing its parameters across iterations. To this end, firstly, we determine the architectural hyperparameters that are beneficial for both types of networks with and without weight sharing (Sections 2.1, 2.3 and 3.1). Secondly, we investigate the effect of multiple outputs on the network training (Sections 2.2 and 3.2). Lastly, we investigate to computational cost of weight sharing (Section 3.3).

We will release the source code upon acceptance for publication.

2 METHODS

In the following, we introduce a network architecture that enables incremental, efficient and accurate semantic segmentation of images. The network processes images of size $x \times y \times 3$ and outputs N semantic maps m_n of size $x \times y \times C$ with C being the number of classes.

2.1 NETWORK ARCHITECTURE

We are interested in network architectures that offer low latency and a small computational footprint facilitating their application in mobile devices. To this end, we design a neural network that heavily re-uses the intermediate activations and potentially also the weights as visualized and described in Figure 2. We consider the memory bandwidth and the number of MACs as meaningful indicators for the computational footprint without restricting ourselves to a specific hardware architecture.

Moreover, since we envision the computation of the iterative block to be efficiently executed en bloc, the size of the *intermediate states* propagated from one to the next iteration is also of high interest. To optimize the network architecture under these objectives (latency, memory bandwidth, number of MACs, size of intermediate state), we choose the number of scales L, the number of iterations N and the number of bottleneck residual blocks K as hyperparameters.

2.2 NETWORK TRAINING

For training, we use a joint cost function for all outputs of the network:

$$\mathcal{L} = \sum_n w_n c_n(\tilde{m}_n, m_n),$$

where c_n is the categorical cross entropy between the true labels $\tilde{m}_n$ and the predictions m_n. The weight factors w_n are normalized as follows: $w_n = w_n/(\sum_i w_i)$.

We use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a learning rate 0.001 that we multiply with 0.1 after 70% and 85% of the number of overall training epochs. We use a batch size of 8 and train the network for 2000 and 600 for the `CamVid` and `Cityscapes` datasets, respectively. If not stated otherwise, we show mean values and errors of the means across 3 trials.

2.3 NETWORK EVALUATION

Throughout this study, we measure the quality of semantic segmentation by calculating the mean intersection-over-union (mIoU Jaccard, 1912), which is the ratio of the area of overlap and the area of union

$$IoU = \frac{label \cap prediction}{label \cup prediction}$$

averaged over all classes.

We consider a network to be good if it has a low latency, which is considered to be proportional to the number of MACs, and high mIoU. To this end, we calculate the area under the curve mIoU (y_n) over MACs (x_n) with output index n as follows:

$$\overline{AUC} = \sum_{n=0}^{N-1} (x_{n+1} - x_n) \left(\frac{(y_{n+1} - y_0) + (y_n - y_0)}{2} \right)$$

with $(x_0, y_0) = (0, 0.00828)$, where y_0 denotes the mIoU at chance level for the `CamVid` dataset. To compensate for different maximum numbers x_N of MACs for different sets of hyperparameters, we normalize as follows: $AUC = \frac{\overline{AUC}}{x_N}$.

If not stated otherwise, all evaluations are done on the average mIoUs over all trials.

2.4 DATASETS

The `CamVid` dataset (Brostow et al., 2008) consists of 701 (367 for training, 101 for validation, 233 for testing) annotated images filmed from a moving car. We use the same static pre-processing as in Badrinarayanan et al. (2017) to obtain images and semantic labels of size 480×360 and normalize the pixel values to the interval $[0, 1]$ by dividing all pixel values by 255. For online data augmentation during training, we horizontally flip the pairs of images and labels at random and use random crops of size 480×352.

The `Cityscapes` dataset (Cordts et al., 2016) consists of 3475 (2975 for training, 500 for validation) annotated images and we use the validation set for testing. We resize the original images and semantic labels to 1024×512 and divide all pixel values by 127.5 and subtract 1 to obtain values in the interval $[-1, 1]$. For online data augmentation during training, we horizontally flip the pairs of images and labels at random.

For both datasets, pixels with class labels not marked for training are ignored in the cost function.

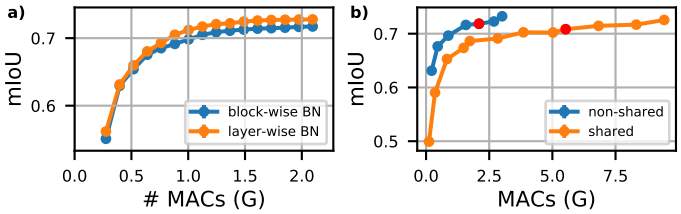

Figure 3: **a)** Comparison of mIoU over MACs between networks with batch normalization applied between iterative building blocks (blue curve) and applied after each convolution (orange curve) on the validation set of the `CamVid` dataset. Note that batch normalization is always applied after each convolution in the data block. Mean values over 5 trials with errors of this mean are shown. **b)** Peak mIoU over MACs for networks with different widths on the validation set of the `CamVid` dataset. Each data point represents the peak mIoU over all network outputs for one specific network width and the corresponding MACs for this output. Mean values over 5 trials with errors of this mean are shown. For the following studies, we selected the networks highlighted in red that have, out of 5 trials, a maximum mIoU of 72.7 and 72.8 for independent and shared weights, respectively.

2.5 Batch normalization

In case of independent parameters between iterations of the iterative building block, batch normalization improves the network training. However, in case of weight sharing, also sharing the parameters of batch normalization between iterations significantly worsens network training. Since not sharing batch-normalization parameters would violate our idea to re-use the identical building block again and again, we place batch-normalization layers between the iterations of the iterative block (see Figure 2b). For comparability, we use the same setup also for networks without weight sharing, although the average validation mIoU is slightly decreased compared to networks that instead use batch normalization after each convolution (Figure 3).

3 Results

In this study, we focus on incremental, efficient and accurate deep neural networks that can be utilized in mobile applications.

3.1 Neural architecture search

In line with Ruiz & Verbeek (2019), we consider the number of multiply-accumulate operations (MACs) as a limiting factor in terms of computational resources. We perform a grid search over the hyperparameters of our network model as described in Figures 1 and 2. For each set of hyperparameters, we choose the largest possible number of channels f that results in a network with less than 2.2 billion MACs for the last output of the network on single samples of the `CamVid` dataset.

Since we are interested in network architectures with high mean intersection-over-union (mIoU) and low latency (number of MACs), we sort the architectures by their area under the curve (see AUC in Section 2.3) for both networks with independent and shared weights. Networks with a peak performance $\max_n(mIoU_n) \leq mIoU_{\text{totmax}} - 0.02$, where $mIoU_{\text{totmax}}$ is the highest peak performance over all sets of hyperparameters, are considered as not suitable for applications and are discarded. For comparability between networks with independent and shared weights, we choose the architecture with the lowest average rank across the two classes of networks and obtain the following set of hyperparameters: $N = 16$ iterations, $L = 3$ scales and $K = 1$ bottleneck residual blocks (for the results of the grid search, see Figure 7 and Figure 6). Note that many iterations ($N = 16$) result in a high re-usage of the iterative building block, a low latency and a low memory footprint of the output feature maps of the iterative block.

Compared to networks with independent weights, networks with shared weights have substantially less free parameters that, consequently, results in a worse mIoU. To compensate for this decrease in peak mIoU we increase the width f of the latter by a factor of 1.76 ($f = 51$ with weight sharing

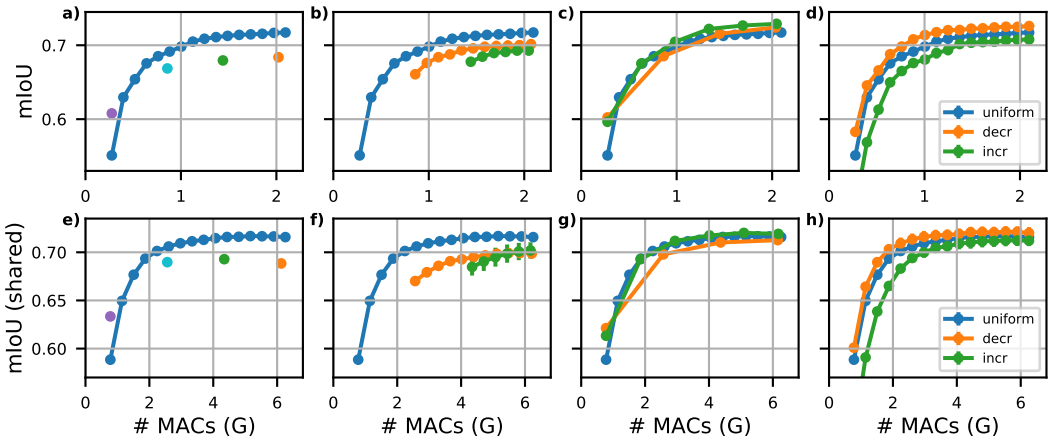

Figure 4: Ablation studies for multi-output training on the validation set of the `CamVid` dataset: The mIoU over MACs is shown for the same network architecture, but for different sets of outputs (different colors in (a-c) and (e-g)). The weight factors w_n are set to zero for removed and to one for remaining outputs. In (d), we linearly modulate the weight factor over the network outputs. We increase the weight factor from 1 to 16 (incr) or decrease it from 16 to 1 (decr). In all panels, mean values over 5 trials with errors of this mean are shown.

compared to $f = 29$ without weight sharing), which corresponds to approximately three times the number of MACs (Figure 3).

3.2 Multi-output training

We study the impact of additional intermediate network outputs on the network performance by training the same network architecture with different sets of weight factors w_n in the loss function as described in Section 2.2. In summary, later network outputs benefit from the additional loss applied to earlier network outputs. This is supported by ablation studies, in which we trained networks with single, only late or thinned out outputs. The following results and conclusions are similar for both network types with and without parameter sharing and, hence, we discuss them jointly.

The training of single outputs is worse than jointly training all outputs, except for only training the first output (see Figure 4a and e). However, a network with only the first output has no practical relevance due to its tiny network size and, consequently, low mIoU.

The intermediate activations of the early layers are optimized for the potentially conflicting tasks of providing the basis for high accuracy at early outputs and, at the same time, top accuracy at late outputs. However, our experiments suggest that especially the optimization of the early outputs improves the performance of late outputs (see Figure 4b and f). The fact that early losses do not conflict with late losses allows for networks with very low latency (see Figure 1b and c). In addition, keeping the important first output, but decreasing the density of outputs, removes unnecessary constraints and slightly improves the mIoU (see Figure 4c and g). The thinned out outputs are likely not required in applications, since the improvement of the mIoU between consecutive outputs is rather small (see Figure 1d).

So far, we removed sets of outputs from the loss function, but kept the weight factors w_n identical over all remaining outputs. For linearly decreasing weight factors over the outputs, i.e. emphasizing early outputs, we observe only minor improvements in the early outputs without degradation in the late outputs (see Figure 4d and h). Surprisingly, linearly increasing the weight factors over the outputs jointly decreases the mIoU for all outputs. The unexpected decrease in the mIoU for late outputs may be attributed to the observed effect that earlier outputs are crucial for late performance as discussed above.

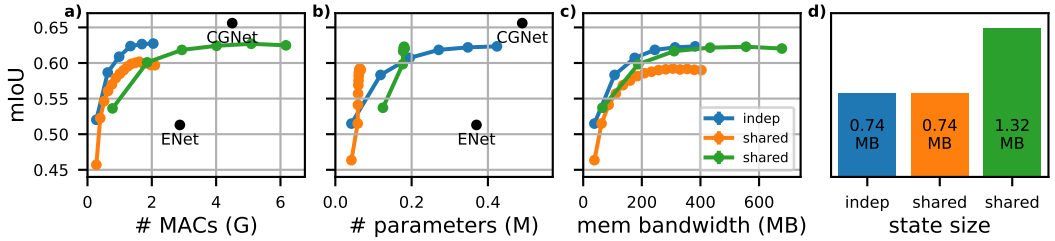

Figure 5: Network performance in terms of mIoU over MACs (a), mIoU over number of parameters (b), mIoU over memory bandwidth (c), and mIoU over memory footprint of the intermediate state (d) on the test set of the `CamVid` dataset. In case of weight sharing, we increased the width of the network (compare orange to green curve; see Section 3.1 for details) to recover to the mIoU of the network with independent weights (blue curve). Out of 3 trials with different random seeds, we show the best sample. According to Figure 4c and g we thin out the losses and, for regularization, use dropout with rate 0.1 after the depth-wise convolutions in the bottleneck residual blocks. Reference data points are taken from Paszke et al. (2016) and Wu et al. (2018). Note that the MACs of these reference data points are normalized to the actual input size (480×360) that was used to obtain the reported mIoU values.

3.3 The cost of weight sharing

In the following, we address the cost of sharing weights in the iterative building block. Specifically, we compare the number of MACs, the number of parameters, the memory bandwidth and the size of the intermediate state between networks with and without weight sharing.

Due to the increased network width to recover the mIoU of the baseline network, networks with weight sharing require 3 times more MACs than baseline networks without weight sharing (see Figure 5a). However, the number of parameters is substantially smaller (44%), since parameters are highly re-used due to weight sharing (see Figure 5b). The *memory bandwidth* is measured in terms of the total memory transfers that are required to compute the network layer by layer and each layer en bloc. In detail, we accumulate the memory requirements for loading the input feature maps and parameters as well as storing the output feature maps across all layers. Since the feature maps have rather large spatial dimensions, they dominate the number of parameters in terms of memory transfers and, hence, the memory bandwidth scales approximately like the width of the network and is 1.77 times higher if weights are shared (see Figure 5c). The same logic applies to the memory footprint of the network's intermediate states (see Section 2.1 and Figure 5c). Note that the memory bandwidth dominates the energy consumption of the system and scales less than the number of MACs.

4 Discussion

This study investigates the trade-off between the homogeneity of highly re-used network building blocks and their required size to keep up with the accuracy of conventional networks. In the context of novel, massively-parallel hardware architectures, we studied two different types of restrictions on the network architecture: First, we re-use the same building block, but update its weights at every iteration. Second, we do not allow such updates, but share the weights between iterations. Summarized, we show that the loss in accuracy induced by these restrictions can be compensated for by widening the networks (see Figure 1b and c). On common datasets for semantic segmentation, both types of restrictions require 3 times more multiply-accumulate operations (MACs) to compensate for their decreasing effect on the network accuracy. In addition, we observe that additional and especially early outputs improve the performance of late outputs, which was also observed by Zhou et al. (2019) on pre-trained networks for age estimation. In principle, the presented methods could also be applied to different tasks, like object classification and detection, for which we expect similar observations.

For the `Cityscapes` dataset, increasing the size of the network with shared weights exceeds the memory of the available GPUs and, hence, we could not show any comparable results on this dataset without modifying the standard training procedure. In parallel to this study, a novel training technique was introduced by Bai et al. (2020) that significantly improves the scaling of vision networks

composed of iterative building blocks with weight sharing. Instead of using backpropagation, they optimize the equilibrium point of a fix-point process described by the iterative execution of the building blocks. They achieved task accuracies comparable to state-of-the-art networks and, to this end, required approximately the same number of parameters compared to the baselines. In line with our study, this results in large iterative building blocks that are executed many times and, consequently, require significantly more MACs than the baseline networks. However, the training method of Bai et al. (2020) can not be applied to networks, for which the weights are allowed to be updated between iterations.

As an extension of this study, partly releasing the constraint of weight sharing may allow for a significant reduction in the required size of the iterative building block. To this end, a ratio between shared and free weights may be chosen close to, but smaller than one. Another interesting open question for further studies is the root cause for the observation that early outputs improve the performance of late outputs. This effect may be attributed to some kind of knowledge distillation and / or to the shortcut for the gradients from network output to input.

REFERENCES

V. Badrinarayanan, A. Kendall, and R. Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2020.

Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV*, pp. 44–57, 2008.

Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. HarDNet: A low memory traffic network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Y. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379, 2016.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113(41):11441–11446, 2016.

Volker Fischer, Jan Koehler, and Thomas Pfeil. The streaming rollout of deep networks - towards fully model-parallel execution. In *Advances in Neural Information Processing Systems 31*, pp. 4039–4050. 2018.

Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *International Conference on Learning Representations (ICLR)*, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018.

Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, 1912.

Tsung-Wei Ke, Michael Maire, and Stella X. Yu. Multigrid neural architectures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Qianli Liao and Tomaso A. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *CoRR*, abs/1604.03640, 2016. URL `http://arxiv.org/abs/1604.03640`.

Sachin Mehta, Mohammad Rastegari, Linda Shapiro, and Hannaneh Hajishirzi. ESPNetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. ENet: A deep neural network architecture for real-time semantic segmentation. *CoRR*, abs/1606.02147, 2016. URL `http://arxiv.org/abs/1606.02147`.

Adria Ruiz and Jakob Verbeek. Adaptative inference cost with convolutional neural mixture models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1947–1950, 2010.

Tianyi Wu, Sheng Tang, Rui Zhang, and Yongdong Zhang. CGNet: A light-weight context guided network for semantic segmentation. *CoRR*, abs/1811.08201, 2018. URL `http://arxiv.org/abs/1811.08201`.

Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J. Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577 (7792):641–646, 2020.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Y. Zhou, Y. Bai, S. S. Bhattacharyya, and H. Huttunen. Elastic neural networks for classification. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 251–255, 2019.

A APPENDIX

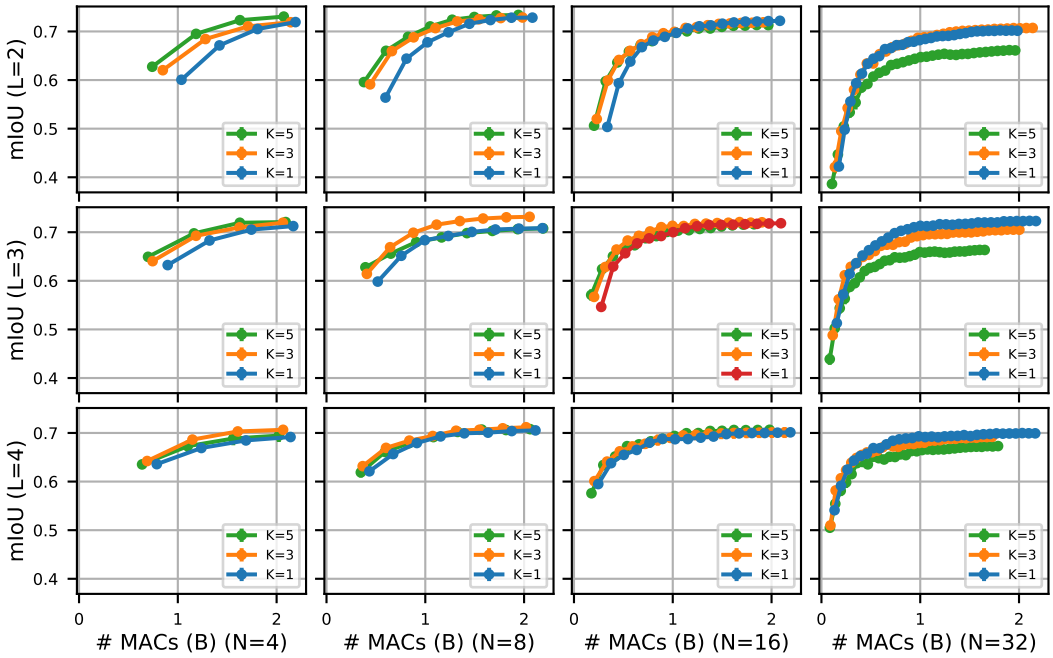

Figure 6: Grid search over the following hyperparameters on the validation set of the `CamVid` dataset: number of layers N, number of blocks L and number of residual blocks K. Parameters are not shared between iterations n of the iterative block. The best set of hyperparameters (for details about the selection, see Section 3.1) is depicted in red.

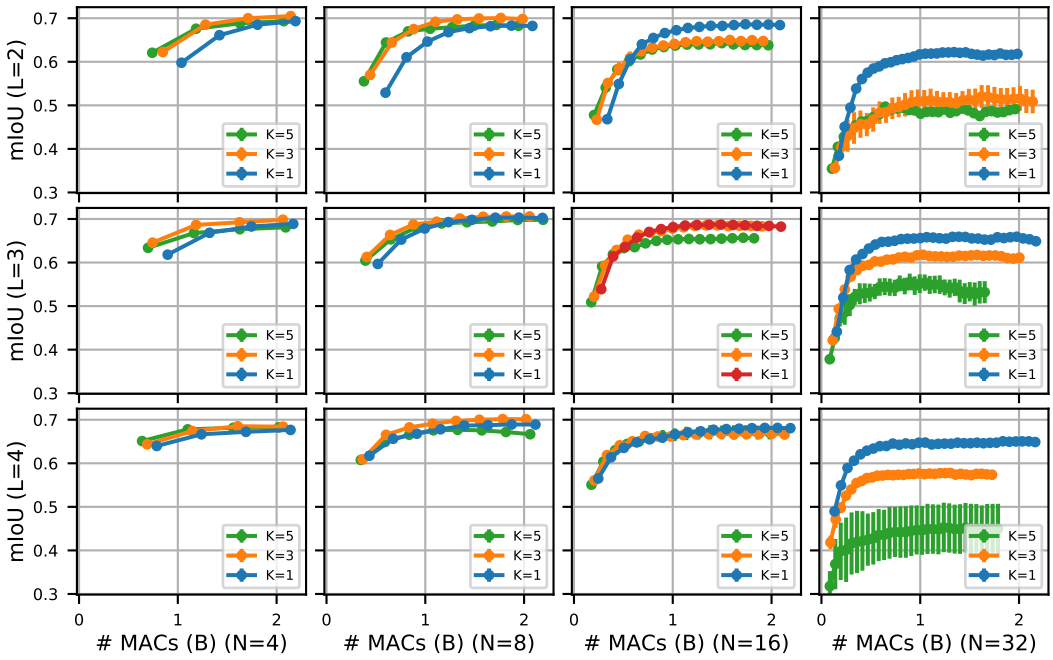

Figure 7: Like Figure 6, but the parameters are shared between iterations n of the iterative block.