PREPRINT for AIAI 2021 - The final publication is available at Springer https://doi.org/10.1007/978-3-030-79150-6

Neural Network Compression Through Shunt Connections and Knowledge Distillation for Semantic Segmentation Problems

Bernhard Haas[1[0000−0003−0379−7281]], Alexander Wendt[1[0000−0002−4909−0006]], Axel Jantsch[1[0000−0003−2251−0004]], and Matthias Wess[1[0000−0002−1877−4114]]

Christian Doppler Laboratory for Embedded Machine Learning, Institute of Computer Technology, TU Vienna, Austria
e1525110@student.tuwien.ac.at, {alexander.wendt, axel.jantsch, matthias.wess}@tuwien.ac.at

Abstract. Employing convolutional neural network models for large scale datasets represents a big challenge. Especially embedded devices with limited resources cannot run most state-of-the-art model architectures in real-time, necessary for many applications. This paper proves the applicability of shunt connections on large scale datasets and narrows this computational gap. Shunt connections is a proposed method for MobileNet compression. We are the first to provide results of shunt connections for the MobileNetV3 model and for segmentation tasks on the Cityscapes dataset, using the DeeplabV3 architecture, on which we achieve compression by 28%, while observing a 3.52 drop in mIoU. The training of shunt-inserted models are optimized through knowledge distillation. The full code used for this work will be available online.

Keywords: Shunt Connections · Knowledge Distillation · Optimization · Latency · Accuracy · CIFAR · Cityscapes · DeepLab · MobileNet · Machine Learning · Embedded Machine Learning

1 Introduction

Compression of deep neural networks is an active field of research. Many state-of-the-art neural network architectures [1, 6, 15] are too computationally expensive to run within set latency requirements on modern embedded hardware. In particular semantic segmentation tasks represent a challenge, since a high image resolution is required to extract high level features, which is computationally expensive. Modern embedded devices like the Nvidia Jetson series and their on-board GPUs have advanced significantly over the last years but real time applications are still challenging.

Neural network compression helps to close this computational gap while still achieving satisfactory accuracy. One proposed method, called shunt connections [14], shows high potential as big parts of a model are replaced by a significantly smaller convolutional model, yielding a considerable speed-up. While in [14] shunt connections were only applied on simple datasets for classification

networks, this work aims to enhance shunt connections for deep neural networks and test them on large scale datasets. In this work, we analyze the number of saved multiply-accumulations and the resulting latency reduction, when applying shunt connections to the MobileNetV3Small-DeeplabV3 model [2, 3, 8] trained on the Cityscapes dataset [4].

We use the following methodology: First, we show that shunt insertion is also applicable to the newer MobileNetV3 model by replicating the original authors' results on the CIFAR datasets [10]. Next, the final fine-tuning step is improved by applying knowledge distillation. Finally, to test for applicability on large scale datasets, we apply shunt connections to compress the MobileNetV3 backbone of the DeeplabV3 architecture trained on the Cityscapes dataset and compare the results against compressing MobileNetV3 using the built-in depth multiplier. Our work contributes to research in the following ways:

- Full information about how to apply shunt connections for easy reproduction
- Improved training of shunt connections through knowledge distillation
- Shunt connections applied to MobileNetV3 trained on the Cityscapes dataset

The full code for inserting shunt connections in residual CNNs will soon be available online[1].

This work does not claim to present the best possible solution regarding shunt connections for a given model and dataset. Instead, it should prove that good results can be achieved for various tasks, including large scale datasets. Hence, we limit ourselves to placing only a single shunt connection into the models and reduce the design space for shunts to two different architectures.

2 Related Work

Shunt connections describe the convolutional neural network compression method of replacing contiguous blocks with a simpler, smaller neural network block. The idea was originally proposed for residual CNNs and motivated by the insight that short paths inside a residual CNN are much more important than long paths [16]. It means that single blocks can be deleted without large accuracy losses. The original paper [14] defines the relative accuracy drop of the block-deleted model as knowledge quotient. By looking at the quotient, one can determine which blocks are easily replaceable by shunt connection.

Shunt architectures are inspired by encoder-decoder models and get trained on extracted feature maps from the original model. The following workflow was proposed by [14]: (1) Train baseline, unmodified model; (2) Calculate knowledge quotients for residual blocks and choose shunt locations; (3) Extract input and output feature maps for shunt training from original model; (4) Choose shunt architecture and train shunt connection; and (5) Place shunt inside original model and fine-tune model.

[1] https://github.com/embedded-machine-learning/ShuntConnector

PREPRINT for AIAI 2021 - The final publication is available at Springer https://doi.org/10.1007/978-3-030-79150-6

In [14], this workflow was applied to the CIFAR [10] and CALTECH[2] datasets. For both datasets the trained networks heavily tend to overfit. Therefore, it is not surprising, that parts of the model can easily be replaced by a simpler model with fewer parameters. More complicated datasets where networks do not tend to overfit, consequently do not contain as much redundancy. Hence, the question appears whether shunt connections also work in those circumstances. Our approach is to apply this concept on a wider scope.

Five different shunt architectures were proposed in the original paper, which consist of blocks similar to the blocks used in the MobileNetV2 model without residual connections. In this work, two of those architectures are used for shunt connections: arch1 and arch4. Arch1 represents a big shunt, consisting of two blocks with a high number of channels, while arch4 consists of a single block with a lower amount of channels. Arch1 is visualized in fig.1, while for the exact architecture of arch4, it is referred to the original paper [14].

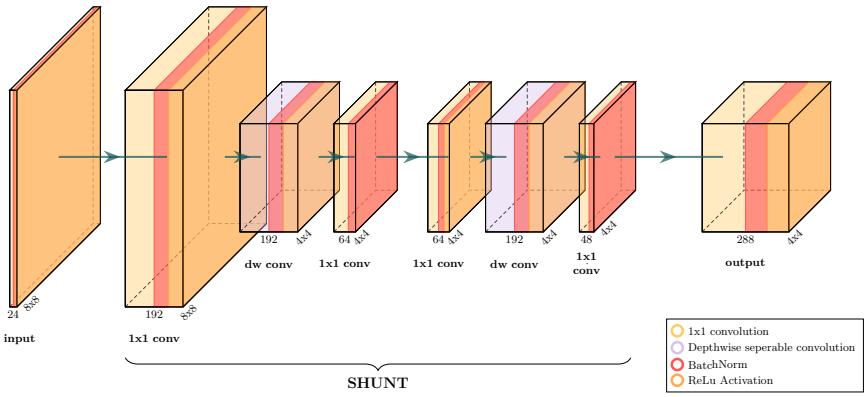


Fig. 1: Shunt architecture used for the CIFAR experiments. Called 'arch1' in [14].[3]

A method that can be used to fine-tune the shunt inserted model is knowledge distillation [7]. It describes the method through which knowledge can be transferred from a big *teacher* model to a smaller *student* model. This way, the student model can achieve better results than without using knowledge distillation.

One possible variant of knowledge distillation is called dark knowledge, which was first proposed in [7] and uses softened targets to transfer information about how similar two classes are. It helps to extract additional information about the dataset from the teacher (t) model, which can be used to train the student (s) model. Let $\mathcal{L}_{CE}(x, y)$ denote the cross-entropy loss between two probability

[2] CALTECH: http://www.vision.caltech.edu/Image$_D$atasets/CaltechPedestrians
[3] Modified from https://github.com/HarisIqbal88/PlotNeuralNet

vectors and sm the softmax activation function which gets applied to the output feature map (out) of a model, then the default loss for our tasks can be defined as

$$\mathcal{L}_D = \mathcal{L}_{CE}(y, \mathtt{sm}(\mathrm{out})). \tag{1}$$

By dividing the output feature map before the softmax by the constant hyperparameter, called temperature, the produced probability distribution is softened up, and similarities between classes become visible. The total loss is then given by:

$$\mathcal{L}_{Total} = \mathcal{L}_D + \lambda \cdot \mathcal{L}_{CE}\left(\mathtt{sm}\left(\frac{\mathrm{out_t}}{T}\right), \mathtt{sm}\left(\frac{\mathrm{out_s}}{T}\right)\right), \tag{2}$$

with λ being a hyperparameter controlling the strength of the distillation loss.

We choose adaptive cross-entropy (ACE) [12] as the distillation method for semantic segmentation tasks since it works similarly to dark knowledge and is simple to implement compared to other approaches [9, 11]. Like other methods, it also takes the produced output from the teacher model and compares them to the student's output. However, this time, the target probability map P will be formed by adding ground truth values $\mathrm{p_{gt}}$ and teacher predictions $\mathrm{p_t}$. Since the teacher model most probably will produce some wrong pixel labels, the teacher's prediction will only be added to pixels, correctly labeled by the teacher model (*). Using the hyperparameter κ to control the strength of distillation, the expanded probability map is given by:

$$P(x,y) = \begin{cases} \kappa \cdot p_t(x,y) + (1-\kappa) \cdot p_{gt}(x,y), & \text{if } (*) \\ p_{gt}(x,y), & \text{otherwise} \end{cases} \tag{3}$$

As we are interested in minimizing latency in embedded systems, MobileNet architectures provide a good base. MobileNetV3 [8] uses a similar architecture as MobileNetV2 [13] with small changes. This architecture was obtained through a network architecture search on the ImageNet dataset [5]. The main addition to its previous version is its use of squeeze-and-excite modules in most residual blocks and hard-swish activation functions instead of standard ReLu. So the standard blocks of the MobileNetV3 consist of a 1x1 expand convolution followed by a 3x3 depthwise-separable convolution with a squeeze-and-excitation module attached and a 1x1 project convolution. Two different model variants: *Small* and *Large*, are defined for MobileNetV3. Additionally, the model can be modified by the depth multiplier parameter, which controls the network's channel numbers.

A custom, lightweight segmentation head for MobileNetV3 was introduced, which architecture is similar to the DeeplabV3+ [3] segmentation head. Low-level features are fed to the Lite Reduced Atrous Spatial Pyramid Pooling (LR-ASPP) module on top of the feature extractor. In contrast, high-level features are extracted by a simple connection followed by a 1x1 convolutional layer. This simple setup allows fast computations with competitive results, especially when combined with the MobileNetV3 as a feature extractor.

3 Generation of Shunt Models

We use the shunt architectures arch1, and arch4 from [14], while small modifications like different channel numbers are made for experiments.

3.1 Shunt Training

The first step of training shunt connections is straight forward. We extract feature maps from the input and the output layer of the shunt connection. Then, we train the shunt as an encoder-decoder to match those outputs for a given input afterward. If feature maps are extracted before training, they must be stored either in memory or on disk. If data augmentation also is applied during feature map extraction, it is much more applicable to do the extraction and training within one step. It means a model gets built, which takes an image as input and calculates the original model's output feature maps and the shunt's output simultaneously. Computing the mean squared loss between those outputs gives us the gradients for updating the shunt's weights.

[14] already showed that a final round of training of the shunt-inserted model is essential to achieve good accuracy. This step is not further explained in that paper, but the name fine-tuning indicates that the model gets partly trained for a small number of epochs and small learning rates. We apply three different setups to analyze the effect of fine-tuning: (1) Train the whole model; (2) Freeze all layers before the shunt connection, and the shunt itself; (3) Apply knowledge distillation while not freezing any layers. Regarding the third option, we use dark knowledge for classification tasks and ACE for segmentation tasks.

For these two training steps, two different learning rate policies are used. The first one is called *poly* and was proposed for the DeeplabV3 architecture in [2]. Poly is used for training baseline models as well as fine-tuning shunt inserted models. Learning rate is reduced by a small amount after every epoch according to equation 4:

$$\mathrm{lr}_{\mathrm{epoch}} = \mathrm{lr}_{\mathrm{base}} \cdot \left(1 - \frac{\mathrm{epoch}}{\mathrm{max\_epochs}}\right)^{\mathrm{power}}, \tag{4}$$

with 'power' being a hyperparameter, which is usually set close to 1. In our case, we use a power = 0.9 for all experiments.

We use another learning strategy for training shunt connections on the feature maps called *plateau*. In our experience, the best way to train shunts is to start with a base learning rate of 0.1 and reduce the learning rate by a factor of 0.1 once the loss of the model reaches a plateau, meaning no improvement for 4 consecutive epochs. We observed that this procedure yields good results independent of shunt architecture and location without any hyperparameter optimization.

3.2 Measuring Performance

Counting multiply-accumulations (MAdds) as a metric of a model is usually used to compare the computational cost of a CNN since they represent the main

contribution to a network's latency. They represent the basic operation in CNNs since MAdds are performed to compute convolutional kernels. Additional to absolute MAdds, we also measure the relative reduction of MAdds for a shunt-inserted model to get a compressed model's theoretical speed-up.

However, as latency is the essential metric for speed on embedded systems and statements through MAdds are limited, we try to measure achieved speed-ups on devices too. Getting meaningful results on embedded devices is not easy, since platform optimizers like NVIDIA TensorRT[4] or Intel OpenVino[5] may not cover all layer types of your model or other problems may occur. Therefore, setting up customized networks on embedded hardware often demands high engineering effort.

Latency estimators like ANNETTE [17] for the Intel NCS2 are handy, since they enable fast testing of compressing techniques. We used this estimator during development of our method and will also report results for the estimated inference speed for DeeplabV3 models.

4 Shunt Architectures on Classification Tasks

We conduct experiments on classification tasks on the small dataset CIFAR10, and CIFAR100 similar to [14]. However, we apply those experiments on the newer MobileNetV3Small and extend them by applying dark knowledge distillation to the fine-tuning step. We use the shunt model and the methodology presented in the previous chapter.

4.1 Datasets

The CIFAR10 [10] dataset consists of 60000 images categorized into 10 classes with 50000 train and 10000 validation images. With an image size of 32x32, it can be trained relatively fast and therefore is used for the first test of our method. The CIFAR100 [10] dataset has the same properties as the other version, expect images are grouped into 100 classes containing 600 images each.

Regarding preprocessing for both datasets, each channel gets standardized independently. Data augmentation for training images is applied through a rotation of images by up to 45 degrees, a maximum shift by up to 6 pixels, and a random horizontal flip.

4.2 Model Setup and Baseline Generation

As a baseline for our experiments, we choose the MobileNetV3Small model with a depth multiplier of 0.5. Original MobileNet architectures are designed with an output stride of 32. Feeding exactly this network with images of CIFAR datasets with an input size of 32x32 would result in feature maps with 1x1

[4] TensorRT: https://developer.nvidia.com/Tensorrt
[5] OpenVino: https://01.org/openvinotoolkit

spatial resolution. Better results are achieved by changing the first two stages of the network not to downsample the image. As a result, we get an output stride of 8, resulting in 4x4 feature maps for the last stage. We run the CIFAR experiments on a single PC using a GTX970, with the software CUDA 10.1 and Tensorflow 2.3 with Keras API is used for all experiments.

The resulting model needs around 6.8 million MAdds. To produce the baseline for all other experiments, the model gets trained for 350 epochs with a batch size of 64 and the poly learning policy with a base learning rate of 1e-2. We use stochastic gradient descent (SGD) with a momentum of 0.9 as the optimizer. A weight decay of 4e-5 is also applied to all layers. The accuracy of the model yields 91.93% for CIFAR10 and 67.10% for CIFAR100.

4.3 Shunt Architecture, Locations and Training

As heuristic to select the location of the shunt connection we calculate the knowledge quotients of blocks with skip connections. Table 1 shows the knowledge quotients of the model for both CIFAR datasets. For blocks without skip connections, the knowledge quotient can not be defined. In the MobileNetV3Small model, these are blocks with stride = 2 or a changing number of input and output channels.

Knowledge quotients for the CIFAR10 datasets are low for every block, which means we can freely choose the shunt location. For the CIFAR100 dataset, knowledge quotients are generally slightly higher, which is expected for a more complex dataset. If possible, block 4 should not be replaced due to its high knowledge quotient.

For maximum effect, we replace blocks 4-10 for CIFAR10 and blocks 5-10 in the case of CIFAR100. It means that the shunt connects two different network stages; hence, it must have a stride 2 layer. It is executed at the start of the shunt, at the first depthwise separable convolution to reduce floating-point operations. Its whole architecture is taken from [14], where it is called architecture 1. The exact setup including channel sizes can be seen in fig.1.

The final step of the procedure is training the shunt-inserted model on the dataset. We found this step crucial to achieving accuracy close to the accuracy of the original model. In [14], this step is called fine-tuning, indicating a low amount of training epochs with very small learning rates. In our experience, higher learning rates may achieve better results. However, we do not want to completely override the learned weights from the original model. The term 'fine-tuning' does not correctly describe this training setup. In the CIFAR datasets, the shunt-inserted model is trained with a batch size of 64 for 300 epochs using the poly learning rate policy with a base learning rate of 1e-3. This setup is used whether or not knowledge distillation is applied.

4.4 Results

Results for both shunt-inserted models and fine-tuned models are summarized in tab.2. Without knowledge distillation, accuracy of the fine-tuned model does

Table 1: Knowledge quotients (KQ) of blocks of the MobileNetV3Small model trained on the CIFAR datasets. Yellow coloured blocks are replaced by the shunt connection.

Block ID	MAdds	KQ CIFAR10	KQ CIFAR100
0	918K	-	-
1	584K	-	-
2	588K	0.12	0.25
3	465K	-	-
4	683K	0.13	0.54
5	683K	0.06	0.16
6	339K	0.02	0.09
7	339K	0.01	0.10
8	400K	-	-
9	599K	0.03	0.24
10	599K	0.03	0.15
Head	567K	-	-
	$\sum$ 6.8M		
		FLOPS REDUCTION	
		$\sim$ 42%	$\sim$ 31%

not reach the one achieved by the original model. Actually, freezing a part of the network to avoid overfitting worsens results. In contrast, dark knowledge helps us achieve almost the original accuracy for CIFAR10 and even better than original accuracy for CIFAR100. This matches the results obtained by [14] for MobileNetV2 on CIFAR, but would not be possible without the addition of knowledge distillation.

5 Shunt Architectures on Segmentation Tasks

Applying shunt connections on fairly large benchmark datasets resembles another challenge, which is investigated in the next section. We show that shunt connections provide competitive results when applied to MobileNetV3 trained on Cityscapes [4] and beats compression using depth multipliers to all metrics.

5.1 Dataset

The Cityscapes [4] contains 19 different classes for semantic segmentation of urban scenes, including cars, pedestrians, vegetation, and cyclists. This dataset is well established for measuring the performance of semantic segmentation models [8], which is why it is chosen for shunt experiments.

It also represents a real-world problem and a much more difficult task than classification on CIFAR datasets. Fine and coarse annotation is available, but

Table 2: Results for CIFAR datasets.

	Acc. CIFAR10 [%]	Acc. CIFAR100 [%]
Original model	91.93	67.10
Shunt-inserted models		
	79.78	53.56
Fine-tuned models with standard shunt		
Standard	88.09	64.63
Freeze	85.66 (-2.43)	60.04 (-4.59)
DK (T=5, λ=2)	**91.36 (+3.27)**	**67.54 (+2.91)**

only fine annotations are used for training. 2975 images are used for training and 500 for validation, normalized between [-1, 1] as preprocessing. Data augmentation is applied through scaling train images with factors between 0.5 and 2.0 and random cropping to 513x1025.

5.2 Model Setup and Baseline Generation

We use our own Keras implementation[6] of the MobileNetV3Small segmentation architecture, proposed in [8]. We verified the model by achieving the same accuracy as the original model from Tensorflow[7] with their pre-trained weights. We tried to replicate the training protocol from [3], which is also referenced by [8]. Due to incomplete details given we achieved only an mIoU of around 10 points lower than in [8].

The best result were achieved by training the model for 700 epochs with batch size 24 and the `poly` learning rate with a base learning rate of 5e-2. We use an output stride (OS) of 8 during training and 32 during evaluation and weight decay of 4e-5 for all layers. The images are validated in full size: 1025x2049, and the ASPP module has not been changed while switching from training to evaluation using 13x13 pooling size with stride 4,5.

All semantic segmentation tasks run on a cluster of eight GTX1080 to have enough graphical memory available. With this setup, we achieve 59.50 mIoU for the Cityscapes dataset.

The knowledge quotients for this baseline model can be seen in tab.3. Note that no result can be reported for the third block since features are extracted from it and fed into the segmentation head. We can see, that no block is particularly hard to replace, which would restrict the decision of possible shunt locations.

[6] https://github.com/embedded-machine-learning/MobileNetV3-Segmentation-Keras
[7] Tensorflow: https://github.com/tensorflow/models/tree/master/research/deeplab

Table 3: Knowledge quotients (KQ) of blocks of the MobileNetV3Small-DeeplabV3 model trained on the Cityscapes datasets.

Block ID	0	1	2	3	4	5	6	7	8	9	10	Head	
MAdds	227 M	53 M	230 M	166 M	128 M	211 M	211 M	114 M	146 M	160 M	75 M	247 M	$\sum$ 2.0 B
KQ	-	-	-	-	0.13	0.15	-	0.20	-	0.20	0.19	-	

5.3 Shunt Architecture, Locations, and Training

Since this section aims to compare shunt connections with the built-in compression technique of MobileNetV3 in the form of depth multipliers, we have to make sure that we achieve similar MAdds reduction. See tab.4 for more information. To do this, we use arch1 and arch4, both proposed by [14]. We also choose two different shunt locations: 4-10 and 5-10. We observed that shunt training is more effective in OS = 32 mode. Therefore, we use it along with the same training procedure as in section 4.3.

Table 4: Results of the MobileNetV3 segmentation model trained on the Cityscapes dataset for different depth multipliers (DM). Percentages in brackets indicate the relative reduction of MAdds and speed-up compared to the baseline model. HS indicates inference on half sized images.

DM	1.0	0.8	0.6
MAdds	196 M	166 M (-15%)	121 M (-38%)
ANNETTE (NCS2)	463 ms	437 ms (-5.6%)	400 ms (-13.6%)
NCS2 (HS)	127 ms	121 ms (-4.7%)	111 ms (-12.6%)
mIoU	59.50	52.73	44.59

Regarding fine-tuning the shunt-inserted model, we observed that training is more effective using OS = 32. Again, we do want to train with quite high learning rates, while not destroying all learned knowledge. Hence we use a base learning rate of 1e-2 for the poly learning policy. We are training for 500 epochs and a batch size of 12. We reduce the batch size here since we need extra memory computing the original model for knowledge distillation, and we want a fair comparison between fine-tuning methods.

5.4 Measuring latency on embedded hardware

Optimization on Intel platforms is done by OpenVINO. We use version 2021.2 and measure inference for half sized images (513x1025) with a single request size on the NCS2. Inference for the full image size gives us an error, which is not further debuggable without big efforts.

Table 5: Results for the Cityscapes datasets. The header describes shunt locations as well as used architectures. The relative reduction of MAdds and speed-up compared to the baseline are written in parentheses. HS indicates inference on half sized images.

	7-10-Arch4	5-10-Arch1	4-10-Arch1
MAdds	166 M (-15%)	141 M (-28%)	120 M (-39%)
ANNETTE (NCS2)	417 ms (-9.9%)	375 ms (-19%)	342 ms (-26.1%)
NCS2 (HS)	111 ms (-12.6%)	98 ms (-22.8%)	92 ms (-27.6%)
Shunt-inserted models - mIoU			
	34.84	37.97	35.13
Fine-tuned models with standard shunt - mIoU			
Standard	**56.52**	54.91	**51.83**
ACE ($\kappa = 0.3$)	56.03 (-0.49)	**55.98 (+1.07)**	51.14 (-0.69)

5.5 Results

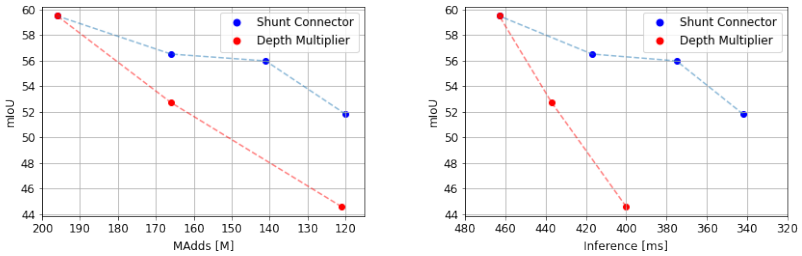
The results of the experiments are summarized in table 5. First of all, one can see that higher MAdd-reduction does not linearly correspond to faster inference. Using shunt connections, around 2/3 of relative MAdd-reduction translates into inference speed-up.

Regarding fine-tuned models, we consider all shunt-connections as successful since the compressed model can achieve relatively high mIoU. Applying knowledge distillation through ACE leads to mixed results. 7-10-arch4 and 4-10-arch1 result in worse results, while 5-10-arch1 gains a boost of one whole point. Knowledge distillation may work well for CIFAR it also acts as strong regularization. Hence, for problems, which tend to overfit, knowledge distillation may have a strong positive effect, while for other problems, like Cityscapes, distillation is less beneficial.

Using the same training procedure as in section 5.2, we achieve the following result for using different depth multipliers for the MobileNetV3 model. See table 4 for results. Note that we make sure, that the channel number of each layer is dividable by 8. Therefore, we cannot observe a smooth curve for model size when lowering the depth multiplier, but rather encounter large steps.

The achieved accuracies and estimated inference speed-ups can be seen in tab.4. One can see that for the depth multiplayer method, only around 1/3 of relative MAdd-reduction translates into speed-up inference.

Fig.2 shows the quantitative comparison between the two methods. This comparison is valid since very similar compression factors are used for both methods: 16% and 38%. We also compare runtimes for half sized images, since the reduction factors only vary slightly with input size. It is visible that shunt-inserted models provide higher mIoU while estimated and measured to running faster on the NCS2 than the models slimmed through depth multipliers. In



(a) mIoU comparison. (b) Estimated inference comparison.

Fig. 2: Comparison between shunt connected models and models obtained through smaller depth multipliers.

conclusion, shunt connections provide faster models, which perform better on Cityscapes, compared to compressed models using low depth multipliers. Hence, shunt-inserted models should always be preferred.

6 Conclusion

It was shown that shunt connections can successfully be applied to various tasks, including classification on CIFAR and semantic segmentation on Cityscapes.

Knowledge distillation in the form of dark knowledge is highly effective for overfitting classification tasks. Applying ACE distillation to large scale segmentation tasks do not provide the same benefit. The increase in performance depends on the exact scenario so that no general statement can be made. Analyzing additional knowledge distillation methods for such problems to get a clearer picture is left as future work.

It is shown that shunt insertion is a competitive method applicable for different tasks and large datasets. Shunt connections should always be preferred over compression using depth multipliers. Especially the actual speedup of the compressed model is convincing.

The problem of finding the optimal shunt location and architecture is still an open problem. Ideally, a systematic, automatable method could be developed. Another open task is also comparing shunt connections with other compression techniques. Especially the comparison between compression rate and translated speed-up seems interesting.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. The computational results presented have been achieved [in part] using the Vienna Scientific Cluster (VSC).

References

1. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection (2020)
2. Chen, L., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation. CoRR **abs/1706.05587** (2017), http://arxiv.org/abs/1706.05587
3. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation (2018)
4. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding (2016)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09 (2009)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385
7. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network (2015)
8. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for mobilenetv3 (2019)
9. Kothandaraman, D., Nambiar, A., Mittal, A.: Domain adaptive knowledge distillation for driving scene semantic segmentation (2020)
10. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
11. Liu, Y., Chen, K., Liu, C., Qin, Z., Luo, Z., Wang, J.: Structured knowledge distillation for semantic segmentation. CoRR **abs/1903.04197** (2019), http://arxiv.org/abs/1903.04197
12. Park, S., Heo, Y.: Knowledge distillation for semantic segmentation using channel and spatial correlations and adaptive cross entropy. Sensors **20**, 4616 (08 2020). https://doi.org/10.3390/s20164616
13. Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. CoRR **abs/1801.04381** (2018), http://arxiv.org/abs/1801.04381
14. Singh, B., Toshniwal, D., Allur, S.K.: Shunt connection: An intelligent skipping of contiguous blocks for optimizing mobilenet-v2. Neural Networks **118**, 192 – 203 (2019). https://doi.org/https://doi.org/10.1016/j.neunet.2019.06.006, http://www.sciencedirect.com/science/article/pii/S0893608019301790
15. Tao, A., Sapra, K., Catanzaro, B.: Hierarchical multi-scale attention for semantic segmentation (2020)
16. Veit, A., Wilber, M., Belongie, S.: Residual networks behave like ensembles of relatively shallow networks (2016)
17. Wess, M., Ivanov, M., Unger, C., Nookala, A., Wendt, A., Jantsch, A.: Annette: Accurate neural network execution time estimation with stacked models. IEEE Access **9**, 3545–3556 (2021). https://doi.org/10.1109/ACCESS.2020.3047259