# Automated Pruning of Neural Networks for Mobile Applications

Andreas Glinserer, Martin Lechner, Alexander Wendt
Christian Doppler Laboratory for Embedded Machine Learning, Institute of Computer Technology, TU Vienna, Austria
Email: e1525864@mail.student.tuwien.ac.at, {martin.lechner, alexander.wendt}@tuwien.ac.at

2021 IEEE 19th International Conference on Industrial Informatics (INDIN) | 978-1-7281-4395-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/INDIN45523.2021.9557525

***Abstract*—Pruning is useful method to compress neural networks and further reduce the required computations and thus the inference speed. This work presents an automatic pruning workflow using an measurement based method to determine which portions of the network only contribute little to the total accuracy. Furthermore to increase the pruneability within networks containing residual blocks this work evaluates zero-padding as an useful complement to existing pruning methods. With zero-padding added to the pruning, we enable the automatic pruning process to also choose layers for pruning which would otherwise not be possible or only possible with removing additional filters which might contribute to the total accuracy. Zero-padding therefore adds the removed channels back into the original output feature map in a manner that the shapes remain identical, but the computations are saved. Using this method we achieved a speedup of up to 21% on CPU based platforms and 5-6% on GPU based execution on a MobileNetV2. The pruned network became comparable to an original network with an applied depth multiplier with only little additional retraining time.**

***Index Terms*—DNN, pruning, Distiller, machine learning, zero-padding, NVIDIA, embedded systems**

## I. Introduction

In recent years, deep neural networks achieve impressive results in various fields ranging from classification tasks and object detection to natural language processing. With these results, also a growth in the used networks is observable. Often, an increase in latency goes hand in hand. Due to the network size, a significant amount of redundancy is included within each network. Model compression aims to reduce redundancy and shrink the model to decrease latency with a minimum loss of accuracy.

Our application is an automatic quality control system, which checks a product for visual impairments. As the product fills a photo, the evaluation happens with a neural network for classification. The control system is running 24/7. The used hardware is a Jetson Xavier AGX[1] because it offers a small form factor and enables processing directly on the machine, thus removing potential network latencies. To speed up inference further, we want to apply model compression.

Our primary target is to speed up the inference and decrease energy consumption while maintaining the current accuracy. To solve this problem, we apply pruning onto a classification network, namely MobileNetV2 [1]. To further increase the prunability of the network, we also want to utilize zero padding to handle dependencies regarding the feature map shape within the network. The main research question is if we can get better results on an NVIDIA Xavier with pruning than by applying the built-in depth multipliers of a MobileNetV2.

We apply the following methodology. We want to use the Distiller [2] framework for pruning and adapt it for usage with MobileNets. Therefore, we first create an iterative pruning approach, in which we test different sets of hyperparameters regarding the following: The amount of pruning and learning rate.

To get a proof of concept, we first apply it on the Cifar10 dataset [3]. It allows us to test quickly and still get acceptable quality. We then apply this approach to networks trained on the ImageNet dataset [4] and prune them. Zero-padding is added to the Distiller framework to enable further pruning to deal with dependencies within the pruned network. Zero-padding is a possible solution in which feature maps between layers are padded back into their original shape after pruning. Finally, we evaluate this approach by comparing it to the depth multipliers of a MobileNetV2. This paper adds the following contributions to the state of the art:

- Implementing zero-padding for easy resolution of pruning dependencies within residual blocks for Distiller
- Evaluating zero-padding as a usable extension to regular pruning methods
- Evaluating the effect of pruning on already optimised networks such as MobileNetV2

## II. Related Work

Pruning has shown in multiple works [5]–[9] to be able to remove significant parts of a network regarding either model size, MACs, or both. One can influence what is optimized by targeting different parts of the network for pruning. For example, convolutional layers are more MAC heavy, while fully connected layers contribute to the model size due to having more parameters than fully connected layers.

The removal of parts of the network to save on computations is not a new concept. It was first introduced by [10] and later on refined by [11]. The recent rise in machine learning and with this the rise of required computational power gave a new surge to finding more efficient networks [1], [12], [13]. MobileNets [1] are widely used networks that offer different

[1]https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-agx-xavier/

configurations of the same network due to the use of depth multipliers.

Pruning can be separated into multiple groups: unstructured and structured pruning. The main difference is that only single points in the network get set to zero or removed with unstructured pruning. Due to mapping most common operations into a matrix multiplication format, this approach may result in a high sparsity, but only with a little speedup. It is possible to utilize this approach by using sparse matrices, but even then, only with high pruning rates of up to 80% for convolution layers, a speedup is reached [14].

To deal with the problems from unstructured pruning, structured pruning uses a similar approach but looks at more coarse units within the matrix. Considering the General Matrix Multiply (GEMM) format, only rows, columns, or groups of columns are removed, which map to channels, partials of the filter kernels, or whole filter kernels, respectively. Due to this, a larger speedup compared to unstructured pruning is reachable.

In Distiller [2], thinning is the process of removing the nodes. The word thinning always refers to removing the network connections, whereas pruning also refers to masking the computations. Since Distiller is designed to be an interactive framework, its parts were modified to be used as a non-interactive and automatic pruning framework. However, this automation comes with a major drawback. Since thinning changes the network structure, some computations within the network are not possible. It affects mainly the point-wise operations between two feature maps, which are common in residual networks. Our approach aims to solve this problem for the Distiller framework. Furthermore, the original Distiller offers a solid base for exploring the network regarding metrics such as accuracy, multiply accumulates, or potential bottlenecks within the architecture regarding memory transfer rates.

The easy approach in pruning is to remove weights that are close to zero due to the assumption that these weights do not hold any viable value to the calculation. The correctness of this assumption is shown in [6], [15], [16].

Han et al. [15] use L1 and L2 regularization to determine which parts are to be removed within the convolution and the fully connected parts of AlexNet. [6] shows the applicability of this ranking approach on multiple convolutional networks. Furthermore, they compare the usage of L1 vs. L2 norm and conclude that there are no notable differences in the applicability regarding the influence on the results. He et al. [16] uses the L2 norm in a soft filter pruning approach.

Recent works on pruning are [5], [7], [17]. They utilize a combined pruning approach removing up to 93% of the nodes within the network, trained on Cifar10. Meng et al. [7] take the structured pruning approach a step further, based on the assumption that filter shape is also essential and introduces filter stripe pruning. This even more fine-grained pruning allows them to remove 50% of the FLOPS in a ResNet18 trained on ImageNet while reducing the accuracy by 0.2%. He et al. [5] utilizes pruning with the help of a reinforcement agent to determine where and how much to prune. This approach reduces 50% of the flops for a ResNet-56 trained on Cifar10 with only a marginal accuracy loss of 0.9% from the original model.

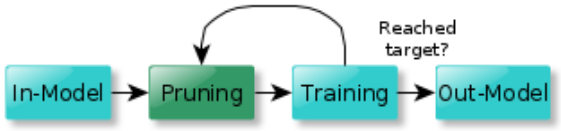


Fig. 1. Iterative Pruning Flow

We want to use the Distiller tool [2] in pruning, as it is currently one of few frameworks to support thinning. At least for networks out-of-the-box and without changing the network definition and creating conversion methods between networks with different dimensions. Problems arise with Distiller when faced with layers that hold complex dependencies, mainly represented thru residual blocks. The main problem and how this was circumvented is described in Section IV-E. Such residual blocks are present within MobileNet, which was our main target for the pruning.

Evaluating what to prune is done in multiple ways. Mainly differentiable in using measurement-based approaches or heuristic methods. Measurement-based approaches record, for example, the activation ratio of neurons and remove those with zero or a low ratio [18]. Other measurement-based approaches use sensitivity analysis [6], [17] to obtain data on how strongly each layer influences the total accuracy. This approach is even further improved by [5] which uses a measurement-based method together with a reinforcement learning agent to generate a pruning plan for a given network.

## III. System Architecture

Based on Distiller [2], we added a non-interactive, automatic and iterative pruning approach. This approach utilizes and builds upon many of the functionalities present within Distiller, especially the thinning functionality to remove the nodes from the network instead of only masking them. To have the automatic pruning process inter-operate seamlessly with different network types, the thinning process was expanded to supply the option to zero-pad outputs in situations where normal thinning would remove big parts due to dependencies.

Iterative pruning is one possibility to prune a given network. Within iterative pruning, the process of removing nodes or filters from the network, the fine-tuning (i.e., retraining), and the evaluation happen interleaved. This process is repeated until a given goal is reached or a condition is fulfilled. Most iterative approaches start with evaluating the network before applying heuristics or getting a better knowledge about where to prune. After this, the network gets retrained and checked if a certain sparsity or evaluation speed is reached. The retraining process can last until a certain accuracy is reached or a given amount of time passed. This pruning is not done with random amounts at random layers. Instead, different methodologies are present to evaluate how much and where to prune. The used approach by us is to use a modified mean value sort to ignore

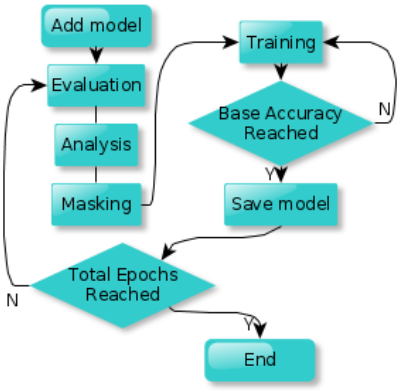


Fig. 2. Implemented Pruning Flow

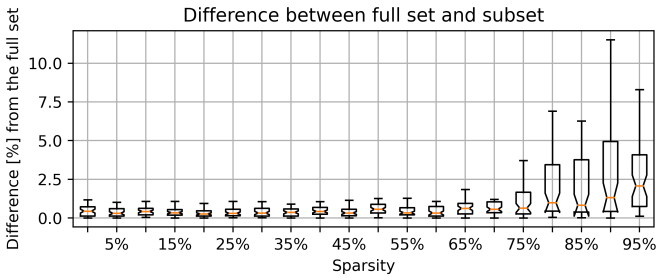


Fig. 3. Difference between using the full set for testing or only a small subset.

smaller layers together with a point selection which depends on the sensitivity gradient.

A basic iterative process is depicted within Figure 1. The node marked in green was expanded from the original Distiller. A model is given, pruned, and retrained. This pruning and retraining happen interleaved. After each training, a given goal or target is checked. In case of fulfilling the goal, the process ends, and the pruned model is returned. If the target is not reached, the model gets pruned further. With the retraining, the loss of information within the network shall be negated. Within the Pruning block, multiple things happen: First, an analysis is done to determine where and how much to prune, the removal itself is done, and evaluation in case of subgoals such as within each step a given amount of calculations shall be removed.

## IV. Implementation

The implemented pruning flow is depicted in detail in Figure 2. In the following sections, the blocks, implementation, and other problems are explained in more detail.

### A. Analysis

Within the analysis block of the iterative pruning process, the model is analyzed by using a sensitivity analysis. Each layer gets pruned step-wise separately, and the influence on the accuracy is measured. Therefore, this process is slower than using heuristics, but due to it being measurement-based holds more accurate values. The base assumption for this process to work is that these individual pruning steps only influence each other little. This assumption holds only true in cases of little to no introduction of sparsity.

In the implementation of the pruning algorithm, a sensitivity analysis similar to [17] and [6] is used to detect which layers hold less value. The L1 norm is used as a basis for this analysis. Within such an analysis, each layer gets pruned independently, and the effect of this gets evaluated by using the dataset. It is becoming a quite time-consuming task, especially for large datasets.

We show that only a small subset of the original data is sufficient to evaluate the effect on the total dataset in our approach. A plot for this difference between using the full validation dataset and only using a tenth of it is shown in Figure 3. As seen on the y-axis, the maximum median error stays around 2% while corresponding to an enormous speed-up, especially for larger datasets. The original evaluation time went down from $\approx$ 60s per layer to $\approx$ 5s for Cifar10. The sparsity on the x-axis is defined as the ratio between the non-masked or later removed against the normal parameters. Therefore a 5% sparsity within a layer is equal to the removal of 5% of the total amount of parameters within this layer. The removal of parameters is also constrained to the filter view onto the weight matrice.

The sensitivity data obtained needs to be sorted to find the best fitting layers for pruning. The search is done automatically by using a sorting method that promotes curves with only a slight degradation in accuracy. A conventional sort for the mean value of the accuracy overall sparsities is not a good fit. This is due to the automatic nature of iterative pruning. A layer might be selected in the first iteration and then again in the second iteration. Therefore, a simple sorting with mean would lead to pruning the same layer more often while leaving other layers untouched. To circumvent this, a simple approach is used. All points within a certain accuracy range along the base accuracy count as promotional points and act as multiplying factors. With this approach, partially pruned layers with only a small portion removed can be selected again. According to this method, also the ranking within the plots is made.

### B. Evaluation

The following steps are executed within the evaluation block: First, the model is evaluated using the test and validation set. Furthermore, measurement data is printed regarding the MACs as well as the number of parameters and sparsity. This evaluation data is also saved. These values are generated using the original functions available within Distiller.

### C. Pruning

This step is named Masking in the pruning flow. The layers to prune are selected within this step and then pruned. The pruning itself happens in a first step by zero-masking the values. This is mostly according to the original Distiller pruning flow. At the end of the process, the model can be taken and thinned separately. We expand the thinning process

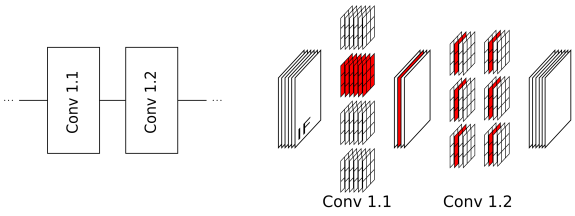


Fig. 4. Pruning a convolution. Removed filter, feature channel and kernel channels in red.

to support zero-padding and add the correct resolving of group wise convolutions dependencies, as described in Section IV-D.

### D. Problems with Pruning

As mentioned in Section III, the structured pruning together with thinning changes the format of the output feature map (OFM) from a layer. This results in dependencies that need to be resolved. Removing a filter from a convolutional layer results in the removal of a channel from the OFM. The OFM from the layer with the removed filter is the input feature map (IFM) for the next layer. Depending on which layer lies afterwards, this leads to different handling of the situation. In the case of a Batch Normalization layer, the corresponding transformation values need to be removed. The BN layer, therefore, transports the dependency to the next layer. In the case of a convolution layer, all filter kernels need to be changed to accommodate the new IFM shape.

To be precise, within each filter kernel, the corresponding channel to the removed kernel from the previous layer needs to be removed. Therefore a convolution layer resolves the dependency due to the OFM having the same shape as the original OFM from this layer. This can be seen in Figure 4. The removed convolution kernel is within Conv 1.1. This kernel would create the second channel in the OFM, which is also the IFM for Conv 1.2. For Conv 1.2 to work, all channels in each filter kernel need to be removed to accommodate the new shape of the feature map.

In the special case of a group-wise convolution, as they are prevalent in MobileNet, the dependency also gets moved on to the next layer. This is due to the special property that each channel of the filter kernel filters only a single channel of the IFM and thus generates a single channel for the OFM. So if one channel is removed from the IFM, this results in the removal of one channel from the filter kernel and thus in the removal of one channel in the OFM.

In the case of residual blocks, the major building blocks of ResNets [19] and also prominent in MobileNets, a further problem is present. In the case of pruning, the last layer before the pointwise addition in such a block, the two OFMs from the separate paths wouldn't fit together anymore. This is depicted in Figure 5. The two feature maps which would be at the pointwise addition do no longer have the same shape $B \times C \times H \times W$ with the batch size, channels, height and width respectively. They differ in the channels.

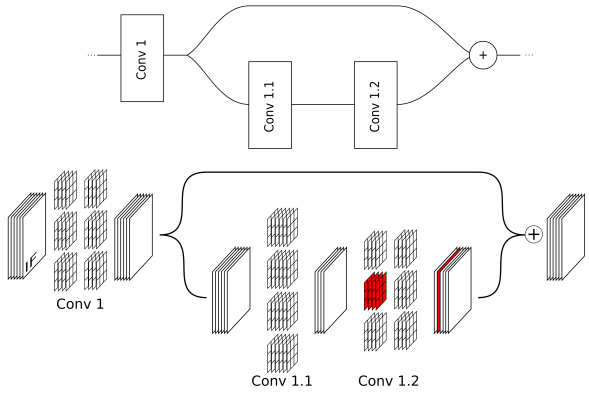


Fig. 5. Pruning problem within a residual block. Removed filter, feature channel and kernel channels in red. The dimensions for the OFM from Conv 1.2 do not fit with the dimension of the OFM from Conv 1.

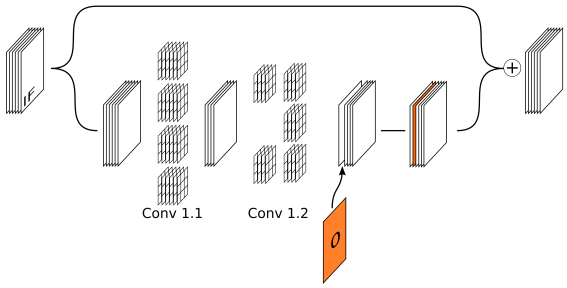


Fig. 6. The residual block as in Figure 5. The dependency between the IFM and the OFM form Conv1.2 is resolved by applying zero-padding.

There are several solutions proposed for this problem. [6] only prunes the layers which would not lead to such a dependency. Thus, it decreases the reachable sparsity. [20] adds additional channel select layers to mask out the insignificant channels. This results in a needed change in the model definition. [8] solves this problem by adding these layers/blocks to the same pruning group. Therefore the layers involved get pruned according to the same parameters.

### E. Zero-Padding

Our approach to deal with the problems described in Section IV-D is zero-padding. It is similar to zero-masking the weights, but with beneficial weight removal. The filter kernel gets pruned appropriately. After the problematic layer, another layer is inserted. This layer then expands the smaller OFM back into its original form. The removed parts of the OFM are filled with zeros and the smaller OFM is mapped into the corresponding channels. This way the OFM now looks as it looked during the masking process. This approach enables the removal of calculations within a wide variety of networks with residual components. The principle is sketched in Figure 6.

### F. Distiller Expansion

We expand the Distiller [2] to automate the pruning selection. Original Distiller was not intended to be used as an

automated framework and rather to be used as a hands-on research tool. Furthermore, the thinning process was expanded to support the aforementioned zero-padding method. The used version did not support the direct thinning of the MobileNetV2 depthwise separable convolutions out-of-the-box. The process was also extended to resolve the dependencies which arise when pruning these blocks of layers.

## V. Results

All measurements were conducted using the MobileNetV2 networks from [21] and only the 1.0 network was taken for pruning. The original network is named according to the depth multiplier, while the pruned network is named accordingly. We selected this network due to its known performance on the hardware. Newer networks, while might being more performant, yield a chance of not being fully supported and therefore not efficiently executable on the target hardware.

TABLE I: MobileNetV2 Measurements and metadata. All this data was given by the source or calculated using the tools within Distiller.

| Network | MACs $\times 10^6$ | Params $\times 10^6$ | Acc [%] |
|---|---|---|---|
| 1.0 | 300.77 | 3.469 | 72.19 |
| 0.75 | 209.08 | 2.636 | 69.98 |
| Pruned | 227.03 | 3.103 | 69.88 |

It can be shown, that even though only a small fraction of the network is pruned, it results in a considerable speedup for MobileNetV2 [1]. In this case, 10.5% of the original network weights are removed. This 10.5% are removed in places such that they equal a reduction of ≈25% in MACs and thus results in a speedup in inference time, such that it becomes comparable to the original implementation used together with a depth multiplier of 0.75 while maintaining a comparable accuracy to the 0.75 networks. Figure 7 shows the positions at which the network was pruned. The accuracy values, the amount of parameters and the multiply accumulates from these networks are listed within Table I. As can be seen, the parameters of the pruned version are roughly in between the 1.0 and 0.75 versions whereas the multiply accumulates are close to the 0.75 version thus showing, those unimportant parameters within convolutional layers were removed. With the presented approach a pruning ratio of 10.5% with regards to the parameters and ≈25% regarding the MACs was reached.

For the evaluation, a Jetson Xavier AGX from NVIDIA was used. Due to it offering a perfect bridge for fully working GPU platforms and embedded systems. Furthermore, NVIDIA offers a TensorRT or TRT[2] optimization for their platform which performs additional optimization on a network graph. These optimizations range from layer fusion to optimal mapping of units onto the GPU. Furthermore, NVIDIA offers different power models for their devices utilizing different amounts of hardware and clock speeds. To switch between these power modes a tool named nvpmodel is present on the Jetson platforms.

[2]https://developer.nvidia.com/tensorrt

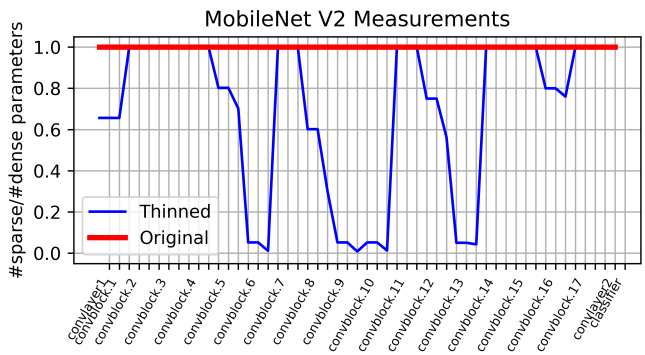


Fig. 7. Graph showing where the automatic pruning removed weights. Each convblock consists of 3 convolution layer as they are implemented within MobileNetV2.

TABLE II: MobileNetV2 inference time measurements with the CPU execution provider.

| Mode | 1.0 [ms] | 0.75 [ms] | Pruned [ms] | Pruned Simplified [ms] |
|---|---|---|---|---|
| m0 JC | 28.41 | 22.85 | 22.43 | 23.42 |
| m0 | 44.56 | 35.94 | 34.14 | 34.01 |
| m2 JC | 67.65 | 52.83 | 52.64 | 55.04 |
| m2 | 69.69 | 54.62 | 54.5 | 57.44 |
| m3 JC | 54.43 | 43.28 | 41.8 | 43.48 |
| m3 | 57.45 | 45.83 | 45.13 | 46.4 |
| m4 JC | 49.39 | 39.17 | 38.44 | 39.77 |
| m4 | 52.58 | 41.56 | 41.28 | 42.9 |
| m5 JC | 44.35 | 34.63 | 34.65 | 36.23 |
| m5 | 48.39 | 38.37 | 38.26 | 40.63 |
| m6 JC | 54.46 | 42.3 | 43.36 | 45.19 |
| m6 | 55.83 | 43.26 | 43.69 | 45.72 |
| m7 JC | 38.25 | 29.78 | 30.23 | 31.35 |
| m7 | 38.05 | 29.78 | 29.89 | 31.29 |

In Figure 8 a comparison between the different execution providers was made: An unpruned MobileNetV2 with a depth multiplier of 1.0, a MobileNetV2 with a depth multiplier of 0.75, the pruned MobileNetV2 and the pruned MobileNetV2 with an onnx simplifier [22]. The simplifier is used to remove orphaned nodes within the ONNX graph. Furthermore, simplification is applied regarding the operations used. Furthermore, static calculations are executed and thus shrink the graph. The measurements were done with jetson clocks activated and the nvpmodel 0 which removes limitations regarding clock speeds. As can be seen, the pruned version executes fastest on the CPU. It performs near equal to the original MobileNetV2 0.75 on the GPU, but loses with TRT when compared to the 0.75 model. The pruned version on which the simplifier [22] was used, is slower than the pure pruned version on both the CPU and the GPU but gives an additional 3% speedup on TRT compared to the pruned version and 10% when compared to the 1.0 version. The reason for this speedup in comparison to the other execution providers was investigated using netron [23]. The speedup is due to the remapping of multiple onnx operations to do the zero-padding into a single more complicated onnx operation. The TensorRT optimiser seems to be more adept in handling simple chains of operations instead of wider operation chains.

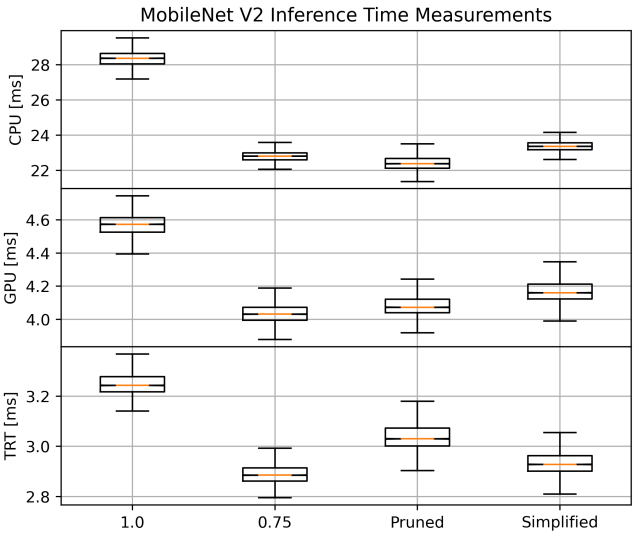


Fig. 8. Inference evaluation on the Jetson Xavier AGX. Measured was the original network, the network trained with a 0.75 depth multiplier, the pruned network and the pruned network simplified using [22].

Within Table II the measurements on the CPU provider are listed. The effect of the different nvpmodels on the inference time is listed. All rows denoted with a JC have jetson clocks activated to maximise the clock speeds. It is visible, that the most speedup is reached by using nvpmodel 0 where a speedup of 36% is reached. The next best speedup is reached for nvpmodel 5 with 9%.

Disregarding the nvpmodel following observations were made:

- The biggest speedup was reached by using the CPU provider. This is probably due to the interruption in the calculation for copying and reconstructing the original feature map.
- The simplifying together with the pruned model always worsens the inference speed or keeps it the same. Except for the TensorRT execution provider.

## VI. Conclusion

The task was to prune a MobileNetV2 to get better performance for the same speedup than by using the depth multiplier of MobileNets. The pruning workflow presented within this work shows that already optimized networks such as MobileNets can be improved further. Our pruning outputs a network comparable in accuracy and inference speed to MobileNetV2 with a depth multiplier of 0.75. However, the advantage of our method is that we need fewer epochs for retraining. Furthermore, our solution evaluates zero-padding as a viable method to assist in pruning networks with more complex dependencies and multiple residual blocks. Thanks to zero-padding all kinds of networks with residual blocks that were previously not pruneable or only partially pruneable, this can now be pruned, thus further expanding the applicability of pruning.

The next steps are to test this method on object detection networks to broaden the application domain and add latency-aware pruning in addition to MAC-aware pruning.

## Acknowledgement

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology, and Development are gratefully acknowledged.

## References

[1] M. Sandler *et al.*, "MobileNetV2: Inverted residuals and linear bottlenecks," 2019. [Online]. Available: http://arxiv.org/abs/1801.04381
[2] N. Zmora *et al.*, "Neural network distiller: A python package for DNN compression research," 2019. [Online]. Available: http://arxiv.org/abs/1910.12232
[3] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.
[4] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.
[5] Y. He *et al.*, "AMC: AutoML for model compression and acceleration on mobile devices," 2019. [Online]. Available: http://arxiv.org/abs/1802.03494
[6] H. Li *et al.*, "Pruning filters for efficient ConvNets," 2017. [Online]. Available: http://arxiv.org/abs/1608.08710
[7] F. Meng *et al.*, "Pruning filter in filter," 2020. [Online]. Available: http://arxiv.org/abs/2009.14410
[8] Z. You *et al.*, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," 2019. [Online]. Available: http://arxiv.org/abs/1909.08174
[9] N. Lee *et al.*, "SNIP: Single-shot network pruning based on connection sensitivity," 2019. [Online]. Available: http://arxiv.org/abs/1810.02340
[10] B. Hassibi *et al.*, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, 1993, pp. 293–299 vol.1.
[11] Y. LeCun *et al.*, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 598–605.
[12] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: http://arxiv.org/abs/1704.04861
[13] S. Mehta *et al.*, "ESPNetv2: A light-weight, power efficient, and general purpose convolutional neural network," 2019. [Online]. Available: http://arxiv.org/abs/1811.11431
[14] J. Yu *et al.*, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 548–560. [Online]. Available: https://dl.acm.org/doi/10.1145/3079856.3080215
[15] S. Han *et al.*, "Learning both weights and connections for efficient neural networks," 2015. [Online]. Available: http://arxiv.org/abs/1506.02626
[16] Y. He *et al.*, "Soft filter pruning for accelerating deep convolutional neural networks," 2018. [Online]. Available: http://arxiv.org/abs/1808.06866
[17] S. Verdenius *et al.*, "Pruning via iterative ranking of sensitivity statistics," 2020. [Online]. Available: http://arxiv.org/abs/2006.00896
[18] H. Hu *et al.*, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016. [Online]. Available: http://arxiv.org/abs/1607.03250
[19] K. He *et al.*, "Deep residual learning for image recognition," 2015. [Online]. Available: http://arxiv.org/abs/1512.03385
[20] Z. Liu *et al.*, "Learning efficient convolutional networks through network slimming," 2017. [Online]. Available: http://arxiv.org/abs/1708.06519
[21] D. Li *et al.*, "Hbonet: Harmonious bottleneck on two orthogonal dimensions," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.
[22] daquexian, "onnx-simplifier." [Online]. Available: https://github.com/daquexian/onnx-simplifier
[23] L. Roeder. Netron. [Online]. Available: https://github.com/lutzroeder/netron