Model Pruning

Patricia Gschoßmann
Technical University Munich
Munich, Germany
patricia.gschossmann@tum.de

March 24, 2021

Abstract

This paper proposes a novel filter pruning framework for structured pruning with predefined target network architectures and without the need for fine-tuning. The main purpose behind this is to encode the input data to solve the problem of data storage, transmission and processing for autonomous vehicles. It can be shown, that directly training the small target model from random initialization can achieve similar performance, as the model obtained from the standard pruning three-stage pipeline. However, the setup used can still be adapted in order to improve results.

1 Introduction

Network Pruning tackles some of the major challenges of deep learning these days: Today's deep learning models are gaining more and more accuracy but are sacrificing fast execution times, low power consumption and storage space in the process.[1] This especially limits the applications for real-time inference and low-resource settings as in autonomous driving. Thus, smaller networks with an optimal number of parameters are more desired than networks with more parameters than necessary.

The typical pruning algorithm is a three-stage pipeline, i.e., training, pruning and fine-tuning (see fig. 1). During pruning, weights are ranked according to a certain criterion (e.g. L1-norm [5]). In each layer, a certain percentage of filters with smaller L1-norm will be pruned while the remaining are kept to best preserve the accuracy. The newly obtained model is retrained to recover the original performance as far as possible. The procedure is stopped, once the performance drops below a certain threshold.
This paper proposes a novel filter pruning framework for structured pruning with predefined target network architectures and without the need for fine-tuning. It can be shown, that directly training the small target model from random initialization can achieve similar performance, as the model obtained from the three-stage pipeline. This shows that the obtained number of parameters is more meaningful than the preserved original weights.

The paper is structured as follows:

- Section 2 describes the structure and procedure of the novel pruning framework proposed in this paper.
- Section 3 provides a summary of previous research on alternative model compression and pruning techniques.
- In section 4 the results of the most successful experiments are presented and evaluated. Additionally, failed attempts are analyzed.
- Finally, the work is summarized in section 5 and an outlook on future research possibilities is given.

[1] https://towardsdatascience.com/pruning-deep-neural-network-56cae1ec5505

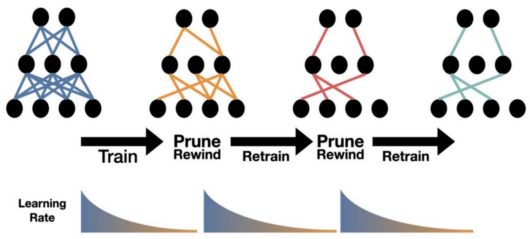


Figure 1: Standard model pruning pipeline (figure taken from [9]).

2 Methodology

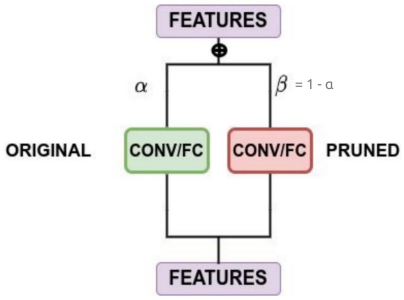


Figure 2: Rough sketch of the new pruning approach (figure taken from [4]). During pruning intermediate model with two separate branches is used, each corresponding to the original model and the smaller target model. The output is the weighted sum of both branches. α is reduced iteratively from 1 to 0.

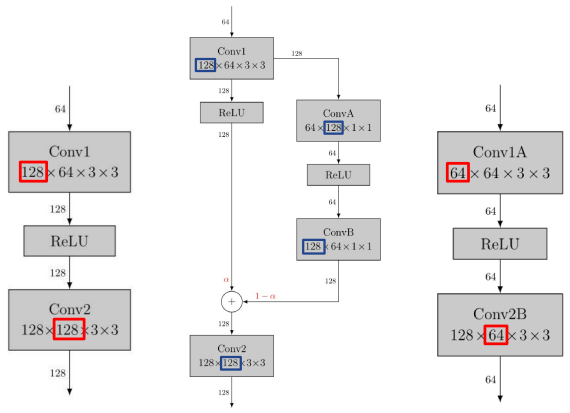
Fig. 2 depicts a high-level sketch of the proposed model pruning approach. The technique aims to directly train a pruned target model without intermediate fine-tuning steps. The key feature of this approach is that the original model with pre-trained weights is preserved during the pruning process and serves as a back-up in case the pruned model's performance drops. At first, an intermediate network is constructed based on the original model's architecture. For each original layer to prune an additional parallel branch is added with the desired number of parameters. The intermediate model's output is the weighted sum of both branches, A weighting factor α is introduced to compute this sum. During pruning α is reduced iteratively from 1 to 0. One question that needs to be answered is how to schedule α.
If successful, the performance should be maintained as good as possible, after reaching $\alpha = 0$. The next step is to shrink the intermediate model, i.e. to remove the original branch. This is done using the so-called Einstein summation[2]. New, smaller weights with the desired number of parameters are obtained.

When constructing a parallel branch, it must be taken into account that no non-linear activation function may occur at the beginning or end in order for this approach to work, as depicted in fig. 3. If done correctly, both layers - the one to be pruned and the first layer in the parallel branch (Conv1 and ConvA in fig. 3) - can be combined into one, as they are only two consecutive linear operations. The same applies to the last layer in the parallel branch and the layer after the one to be pruned (ConvB and Conv2 in fig. 3).
Furthermore, the parallel branch's architecture is dependent on the type of layers occurring in the original model, since the weights of two different layers cannot be mathematically combined with each other. Fig. 3 shows an exemplary application of the framework on two consecutive convolutional layers.

3 Related Work

Luo et al. [8] prune a channel based on the outputs of its next layer, not its own. In more detail, the channels that have the smallest effects on the next layer's activation values are removed.
Similarly, He et al. [3] propose an regression based channel selection and least squares reconstruction technique that prunes each layer by minimizing the feature map reconstruction error of the next layer.
Liu et al. [7] use a method called network slimming to identify unimportant channels during training. This is done by multiplying the output of each channel with an individual scaling factor γ. The network weights and scaling factors are trained, with sparsity regularization imposed on the latter. Channels with small factors are re-

[2] https://pytorch.org/docs/stable/generated/torch.einsum.html



(a) Original model. Output and input channels marked in red should be reduced.

(b) Intermediate model with parallel branches. Output and input channels marked in blue are merged through the weight multiplication.

(c) Pruned model after multiplication. Reduced output and input channels are marked in red.

Figure 3: Detailed view of the new pruning approach on two consecutive convolutional layers. For each original layer to prune an additional parallel branch is added with the desired number of parameters. After reaching $\alpha = 0$ the original branch is removed by multiplying Conv1 with ConvA and ConvB with Conv2 using the so-called Einstein summation[3]. New, smaller weights with the desired number of parameters are obtained.

moved followed by fine-tuning the pruned network. They were able to decrease the computational cost of state-of-the-art networks up to 20× with no accuracy loss.

The Multi-Dimensional-Pruning framework from Tang et al. [11] follows a similar approach as the one used in this paper. The framework aims to simultaneously compress CNNs on multiple dimensions by constructing an over-parameterized network from a given original network to be pruned: Each convolutional layer is expanded into multiple parallel branches. After training this over-parameterized network, the unimportant branches and channels are pruned. A fine-tuning stage follows to recover from the accuracy drop.

In [6] important weights are identified throughout training. Similarly to this paper, a closely related dense model provides a feedback signal to correct pruning errors during training.

AutoML for Model Compression (AMC) [2] automatically learns the compression ratio of each layer through reinforcement learning.

Frankle et al. [1] hypothesize, that every randomly-initialized, dense neural network contains a sparse sub-network (the ”winning ticket”) for every desired sparsity level that can be trained to the same or better performance as the dense model. They propose an algorithm to identify such winning tickets.

Wang et al. [12] observed, that pruning on pre-trained weights reduced the search space for the pruned structure. They propose a new network pruning pipeline which directly learns a pruned network structure without relying on pre-trained weights. The pruned model weights are learned from scratch.

4 Experiments and Evaluation

I evaluated this approach for the popular VGG-16 on the CIFAR-10 dataset as well as on an RGB auto-encoder on images from the CARLA environment [4].

4.1 Experiments with VGG-16

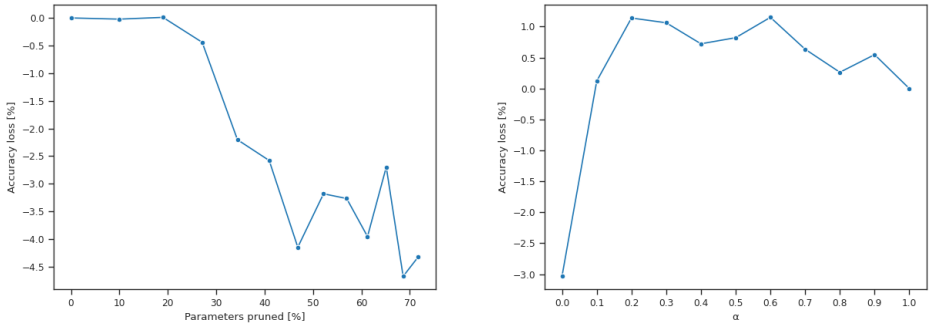
The VGG-16 [10] is a 16 layer neural network, with 13 convolutional and three fully-connected layers. Kuangliu's slightly modified implementation[5] served as a baseline for this task. He combined the last three linear layers of the original model to only one.

Fig. 4a shows the percentage of parameters pruned vs. the drop in accuracy using standard pruning. The original model reached a validation accuracy of 92%. This value did not fall below the threshold of -4% until the sixth pruning step, where the network was already reduced to just above 50% of its original size. As a final result standard pruning was able to reduce the model to ≈35% of its original size and simultaneously maintain a validation accuracy of nearly 90%.

My goal for this project was to prune those ≈65% (more accurately: 65,625%) of parameters all at once. This is done by reducing the number of output channels in

[4] https://carla.org/

[5] `https://github.com/kuangliu/pytorch-cifar`



(a) Standard model pruning

(b) New pruning approach

Figure 4: Pruning effects on the validation accuracy of a VGG-16 on the CIFAR-10 dataset.

each convolution by the required amount (e.g. 64 output channels are reduced to 22).

4.1.1 Challenges

In the following, various approaches that failed in the beginning of this project are listed.

For the approach to work, the learning rate must not be too high - as with all other machine learning tasks. Since I trained the original VGG with an initial learning rate of 0.1 and a learning rate scheduler, which reduces the learning rate on a plateau, I decided to use the same set up during pruning. However, one crucial mistake was to add an early-stopping callback in combination with the LR-scheduler. As a result, intermediate α-iterations stopped too early, causing the validation accuracy to drop to 81% at $\alpha = 0.3$. The model was not able to recover from that, as depicted in fig. 8 in the appendix.
Before learning rate and callback could be identified as the source of error, the simple α-step-scheduler was replaced by an multiplicative decay. I assumed furthermore that the model was overfitting on the training dataset. One attempt to fix the problem was to use data augmentation using Gaussian noise. Unfortunately, none of these approaches led to improvements (see appendix fig. 9).
Another approach I pursued to maintain the validation accuracy was training the model as in a regression task. This was done by using the logits of the pre-trained model as target values. However, further investigations in this direction were aborted after other approaches showed more promising results.
Allowing weight updates for the pre-trained model branch was definitely the decisive change to make the new pruning approach work. Enabling it for α close to zero alone achieved significant improvements. The best results, however, could be obtained by allowing weight updates from the beginning. In this way, it is not necessary to use an exponential α-decay - a step-scheduler was sufficient enough. The results are presented and evaluated in the following section.

4.1.2 Results

Following results were obtained by allowing weight updates for the pre-trained model branch from start. α was reduced using a step-schedule with a deacy-rate of 0.1. All intermediate models reached their best performance within a maximum number of 35 epochs. With an initial learning rate of 0.001 for each iteration, the model was able to maintain a validation accuracy of 92% throughout the whole training, until $\alpha = 0.0$ (see fig. 4b). The accuracy could even be increased a little for some levels. The accuracy dropped by 6% at $\alpha = 0.0$. Resuming training two times, first with a learning rate of 0.01, second with a learing rate of 0.1, lead to a final validation accuracy of 88.5% In total, the model was

trained for 74 epochs at $\alpha =$.
Fig. 4b summarizes the overall development of the test accuracy at each α.

After the final model reached the desired accuracy, its parallel branches were pruned to obtain the smaller version of it. The resulting pruned model was again tested on the test data, to ensure that the pruning was done correctly. It maintained the same accuracy as during the validation. Additional tests regarding time and average memory consumption for the pruned model were executed in order to compare the performance of the original and pruned model. The results are listed in table 1. On the GPU it can be observed that the smaller model takes just as much time per image as the original model, while significantly fewer GPU resources are required. The result is not surprising, since all multiplications in a layer are executed simultaneously on a GPU, i.e. fewer multiplications due to fewer parameters (with the same number of layers) have no effect on the time. However, since in general fewer multiplications are computed, fewer resources are used. This is different on the CPU, which is not able to run all multiplications of a layer at the same time. As a result, the smaller model is obviously able to classify an image more quickly than the original model while consuming fewer resources at the same time.

	Original	Pruned
Time/img (GPU)	$\approx 0.0420s$	≈ 0.0416
GPU consumption	2.00GB	1.00GB
Time/img (CPU)	$\approx 0.0193s$	$\approx 0.0060s$
RAM consumption	3.99GB	3.78GB

Table 1: Comparison of original and pruned VGG-16 on GPU and CPU.

4.2 Experiments with the RGB auto-encoder

The original model's architecture is depicted in table 2 in the appendix. It reduces input images of size $128 \times 128 \times 3$ to $16 \times 16 \times 128$ pixels. A weighted average of L1 and SSIM[6] is used as training objective. Example reconstruction results are depicted in the figures 7b and 7d.

The original model reached a validation loss of 1.40. As with the VGG, my goal was to prune 65% of the model parameters, leading to a bottleneck of size $44 \times 16 \times 16$. However, it is unknown what results could already been achieved under such conditions, since standard pruning was not applied previously. Because of this, a comparison between standard pruning and the new approach is a little more difficult.

Furthermore, in the beginning it was unclear if the decoder could be pruned at the same time while the latent space was changing (due to the reduced bottleneck size). I therefore investigated two different approaches to prune the auto-encoder:

- Prune encoder and decoder sequentially
- Prune en- and decoder simultaneously

For both approaches the same pruning setup as with the VGG was used.

4.2.1 Challenges

Since a working setup was already found when pruning the VGG, there were hardly any unsuccessful attempts for following experiments. However, an implementation error led to the assumption that pruning the encoder pruning failed after setting $\alpha = 0$. To avoid this, the scheduler was changed to an exponential decay and weight decay was added with a decay-rate of 1e-4. The model did not show any any improvements with this approach, as depicted in fig. 10 in the appendix. Fortunately, the mistake was discovered immediately afterwards, which is why the approach was not further pursued. Regardless of that, it would be interesting to test weight decay with a bigger decay-rate in further experiments.

4.2.2 Results

Following results were obtained by allowing weight updates for the pre-trained model branch from start. α was

[6] https://github.com/Po-Hsun-Su/pytorch-ssim

reduced using a step-schedule with a deacy-rate of 0.1. All intermediate models reached their best performance within a maximum number of 60 epochs. As before - following the approach of cosine-annealing LR-schedulers - resuming training with higher learning rates helped to overcome local minima. In both approaches, the models at $\alpha =$ were thus trained for over 100 epochs.

Fig. 6 summarizes how the model's performance developed for both approaches. Unlike with the VGG, the model's performance dropped immediately during the α-schedule, regardless of the approach - it was certainly not possible to achieve any improvement in the meantime. What is similar, though, is that until $\alpha = 0$, both models could maintain a rather stable performance, which dropped significantly at $\alpha = 0$. Until then, it seems that it makes no difference which of the two approaches is chosen. However, separate pruning resulted overall in a greater performance drop: Since setting $\alpha = 0$ decreased performance every time, the final model after pruning en- and decoder at the same time is better. In terms of pruning duration and final performance, it is therefore recommended to prune en- and decoder simultaneously.

Fig. 7 shows final reconstructed outputs of both pruned models using two examples. As expected knowing the final validation loss of both approaches, the images are much better reconstructed when pruning the en- and decoder at the same time. It is worth to mention that the reconstructed images after multiplication do not exactly match those before multiplying the weights, indicating that there is a small error in the implementation. Unfortunately this could not be fixed.

Both pruned models were also tested regarding time and average memory consumption. Fig. 5 lists the results of both approaches. In this experiment, too, the advantages of pruning for the GPU are clearly visible: The GPU consumption has been reduced by more than a half. As with the VGG16, the advantages on the CPU can be particularly observed in the time per image. Nevertheless, a small improvement in terms of RAM consumption is also visible. However, this cannot be compared with the improvement on the GPU. Overall, it can be said that, there are no discernible differences between both final models regarding time and memory consumption. The result is not surprising, since the number of final parameters is the same.

	Original	Pruned
Time/img (GPU)	$\approx 0.0531s$	$\approx 0.0532s$
GPU consumption	7.18GB	3.24GB
Time/img (CPU)	$\approx 0.0858s$	$\approx 0.0509s$
RAM consumption	7.98GB	7.87GB

(a) En- and decoder were pruned separately.

	Original	Pruned
Time/img (GPU)	$\approx 0.0531s$	$\approx 0.531s$
GPU consumption	8.21GB	3.59GB
Time/img (CPU)	$\approx 0.0876s$	$\approx 0.0522s$
RAM consumption	7.98GB	7.87GB

(b) En- and decoder were pruned simultaneously.

Figure 5: Comparison of original and pruned RGB autoencoder on GPU and CPU.

5 Conclusion

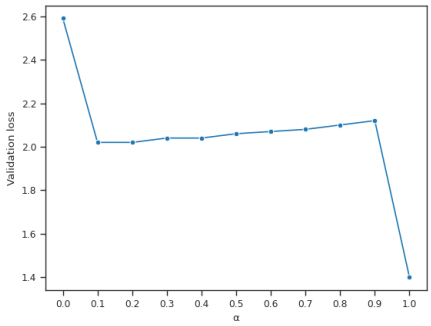
5.1 Summary

The goal of this paper was to prune different neural networks using a novel filter pruning framework for structured pruning with predefined target network architectures and without the need for fine-tuning. It could be shown, that directly training the small target model from random initialization can achieve similar performance, as the model obtained from the standard pruning three-stage pipeline. The best results have been achieved by allowing weight updates for the pre-trained model branch and increasing the learning rate as soon as local minima occur[7]. In all successful experiments an step-schedule with a step-size of 0.1 was used to schedule α. Other schedulers did not produce better results, indicating that the learning rate contributes much more to the success of this framework than α. Nonetheless, the question of the ideal schedule remains open.

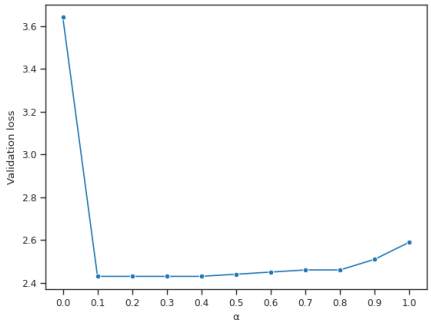
5.2 Outlook

In order to answer open questions, the next logical step would be to execute both experiments with a larger step

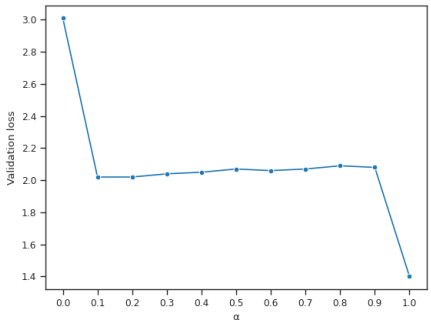
[7] Which is likely to happen once α is set equal to zero.



(a) Pruning of encoder



(b) Pruning of decoder (after pruning of encoder)



(c) Pruning en- and decoder

Figure 6: Pruning effects on the validation loss of the RGB auto-encoder on CARLA images.

size. Overall, it would be desirable if one could use the proposed approach to set α directly or very quickly to 0.1 or a similarly low value, to then continue with an exponential schedule, for example. I am optimistic that something similar would be possible, since the intermediate models' performance did not change much during pruning in both experiments.

It would also make sense to further examine effects of so far unsuccessful attempts such as weight decay with a different decay-rate, though they were not further pursued, but could also help to achieve better results.

In parallel, the proposed pruning framework could be applied to other, more complex models, such as an auto-encoder with skip connections. Another, more sophisticated network would be a DCGAN, whose architecture is similar to that of an auto-encoder. Here, too, it would be interesting whether pruning the generator and discriminator separately or simultaneously makes a significant difference.

(a) Original image

(b) Reconstructed image before pruning

(c) Original image

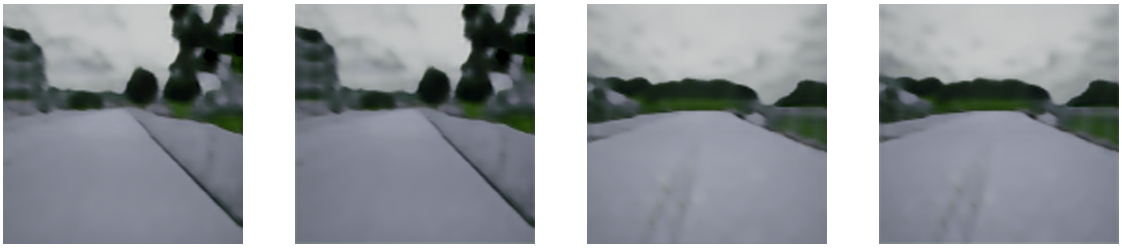
(d) Reconstructed image before pruning

(e) Reconstructed image after pruning and before multiplication of weights (sim.)

(f) Reconstructed image after pruning and after multiplication of weights (sim.)

(g) Reconstructed image after pruning and before multiplication of weights (sim.)

(h) Reconstructed image after pruning and after multiplication of weights (sim.)



(i) Reconstructed image after pruning and before multiplication of weights (seq.)

(j) Reconstructed image after pruning and after multiplication of weights (seq.)

(k) Reconstructed image after pruning and before multiplication of weights (seq.)

(l) Reconstructed image after pruning and after multiplication of weights (seq.)

Figure 7: Final results after pruning both, encoder and decoder of the RGB auto-encoder. First row shows original image and unpruned auto-encoder results. Second row shows results of pruning en- and decoder simultaneously (sim.). Third row shows results of pruning en- and decoder sequentially (seq.).

References

[1] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv*, pages 1–42, 2018.

[2] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11211 LNCS:815–832, 2018.

[3] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *arXiv*, pages 1389–1397, 2017.

[4] Qadeer Khan. Learning For Self-Driving Cars and Intelligent Systems P1 : Model Pruning. (December), 2020.

[5] Hao Li, Hanan Samet, Asim Kadav, Igor Durdanovic, and Hans Peter Graf. Pruning filters for efficient convnets. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, (2016):1–13, 2017.

[6] Tao Lin, Luis Barba, Sebastian U. Stich, Daniil Dmitriev, and Martin Jaggi. Dynamic Model Pruning With Feedback. *arXiv*, 2020.

[7] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient Convolutional Networks through Network Slimming. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:2755–2763, 2017.

[8] Jian Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:5068–5076, 2017.

[9] Kim Martineau. A foolproof way to shrink deep learning models. `https://news.mit.edu/2020/foolproof-way-shrink-deep-learning-models-0430`, 2020. [Online; accessed 20-March-2021].

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

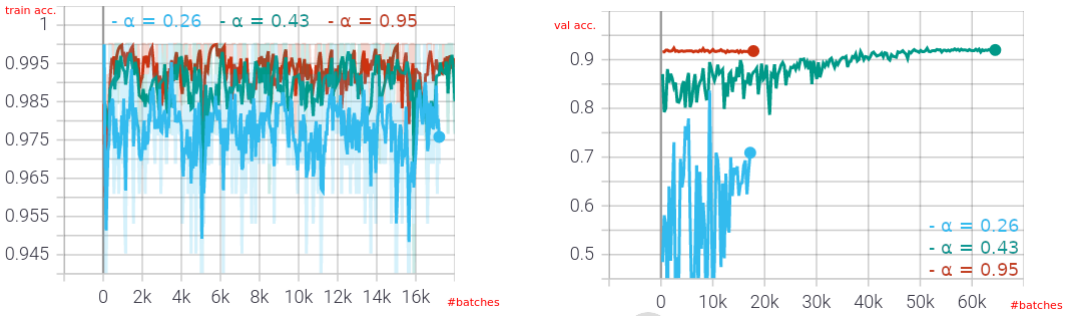
[11] Yajiao Tang, Junkai Ji, Shangce Gao, Hongwei Dai, Yang Yu, and Yuki Todo. A Pruning Neural Network Model in Credit Classification Analysis. *Computational intelligence and neuroscience*, 2018:9390410, 2018.

[12] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. *arXiv*, 2019.

A Appendix

RGB Auto-encoder		
Name	Type	Weight Shape
Conv0	Conv2d	$64 \times 3 \times 3 \times 3$
Conv1	Conv2d	$64 \times 64 \times 3 \times 3$
max_pooling	MaxPool2d	-
Conv2	Conv2d	$128 \times 64 \times 3 \times 3$
Conv3	Conv2d	$128 \times 128 \times 3 \times 3$
max_pooling	MaxPool2d	-
Conv4	Conv2d	$128 \times 128 \times 3 \times 3$
Conv5	Conv2d	$128 \times 128 \times 3 \times 3$
Conv6	Conv2d	$128 \times 128 \times 3 \times 3$
max_pooling	MaxPool2d	-
bottleneck		
ConvTrans0	ConvTranspose2d	$128 \times 128 \times 2 \times 2$
Conv7	Conv2d	$128 \times 128 \times 3 \times 3$
Conv8	Conv2d	$128 \times 128 \times 3 \times 3$
Conv9	Conv2d	$128 \times 128 \times 3 \times 3$
ConvTrans1	ConvTranspose2d	$128 \times 128 \times 2 \times 2$
Conv10	Conv2d	$128 \times 128 \times 3 \times 3$
Conv11	Conv2d	$128 \times 64 \times 3 \times 3$
ConvTrans2	ConvTranspose2d	$64 \times 64 \times 2 \times 2$
Conv12	Conv2d	$64 \times 64 \times 3 \times 3$
Conv13	Conv2d	$64 \times 3 \times 3 \times 3$

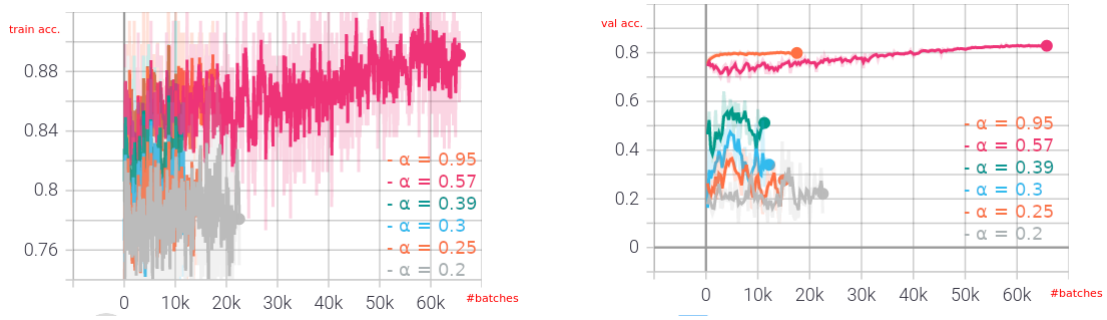
Table 2: Original RGB auto-encoder architecture. Each convolutional layer is followed by a BatchNorm and a ReLU activation.



(a) Training accuracy

(b) Validation accuracy

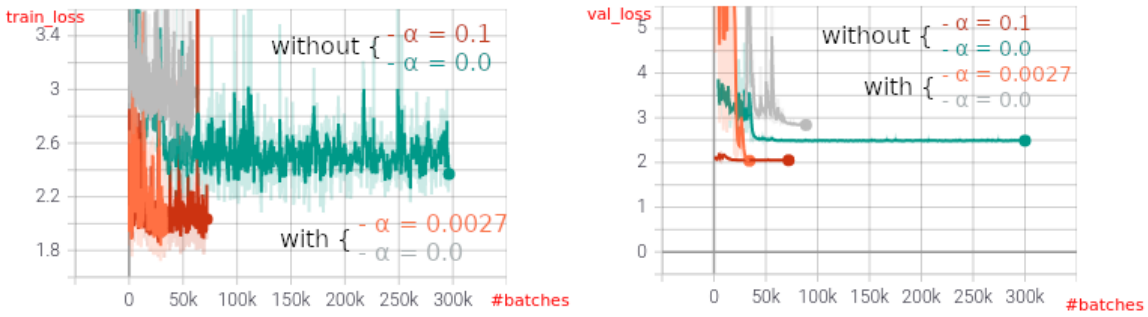
Figure 8: Accuracy development for selected α-values using an exponential schedule with a decay-rate of 0.05.



(a) Training accuracy

(b) Validation accuracy

Figure 9: Accuracy development for selected α-values with data augmentation applied on half of the training data.



(a) Training accuracy

(b) Validation accuracy

Figure 10: Accuracy development without weight decay (using a step decay) and with weight decay (using a weight decay-rate of 1e-4 and an exponential decay).