RESCIENCE C

Replication / ML Reproducibility Challenge 2021

[Re] Reproducibility Study: Comparing Rewinding and Fine-tuning in Neural Network Pruning

Szymon Mikler[1, ID]

[1]University of Wroclaw, Poland

Edited by
(Editor)

Received
–

Published
–

DOI
10.0000/zenodo.0000000

1 Reproducibility Summary

Scope of Reproducibility

We are reproducing *Comparing Rewinding and Fine-tuning in Neural Networks*, by [1]. In this work the authors compare three different approaches to retraining neural networks after pruning: 1) fine-tuning, 2) rewinding weights as in [2] and 3) a new, original method involving learning rate rewinding, building upon [2]. We reproduce the results of all three approaches, but we focus on verifying Renda's original approach: learning rate rewinding, since it is newly proposed and is described as a universal alternative to other methods.
As the authors of [1], we used CIFAR10 for most of the experiments, but we added experiments on a larger version of this dataset: CIFAR100. We have also extended the list of tested network architectures to include Wide ResNets [3]. The new experiments led us to discover the limitations of learning rate rewinding which in some cases can worsen pruning results on large neural network architectures.

Methodology

We implemented the code ourselves in Python with TensorFlow 2, basing our implementation of the paper alone and without consulting the source code provided by the authors. We ran two sets of experiments. In the reproduction set, we have striven to exactly reproduce the experimental conditions of [1]. We have also conducted additional experiments, which use other network architectures, effectively showing results previously unreported by the authors. We did not cover all originally reported experiments – we covered as many as needed to state the validity of claims. We used Google Cloud resources and a local machine with 2x RTX 3080 GPUs.

Results

We were able to reproduce the exact results reported by the authors in all originally reported scenarios. However, extended results on larger Wide Residual Networks have demonstrated the limitations of the newly proposed learning rate rewinding – we observed a previously unreported accuracy degradation in low sparsity ranges. Nevertheless, the general conclusion of the paper still holds and was indeed reproduced.

Correspondence should be addressed to Szymon Mikler (sjmikler@gmail.com)
The authors have declared that no competing interests exist.
Code is available at https://github.com/gahaalt/reproducing-comparing-rewinding-and-finetuning – DOI 10.5281/zenodo.6519109. – SWH swh:1:dir:886a4c9a0bdecdbf65f2cab3ae7404a6796bc451.
Open peer review is available at https://openreview.net/forum?id=HxWEL2zQ3AK.

What was easy

Re-implementation of pruning and retraining methods was technically easy, as it is based on a popular and simple pruning criterion – magnitude pruning. Original work was descriptive enough to reproduce the results with satisfying results without consulting the code.

What was difficult

Not every design choice was mentioned in the paper, thus reproducing the exact results was rather difficult and required a meticulous choice of hyper-parameters. Experiments on ImageNet and WMT16 datasets were time consuming and required extensive resources, thus we did not verify them.

Communication with original authors

We did not consult the original authors, as there was no need to.

2 Introduction

Neural network pruning is an algorithm that intends to decrease the size of a network, usually by removing some of its connections or setting their weights to 0. This procedure generally allows obtaining smaller and more efficient models. It often turns out that these smaller networks are as accurate as their bigger counterparts or the accuracy loss is negligible. A common way to obtain such high quality sparse network is to prune it after the training has finished [2], [4]. Networks that have already converged are easier to prune than randomly initialized networks [5], [4]. After pruning, more training is usually required to restore the lost accuracy. Although there are a few ways to retrain the network, finetuning might be the easiest and most often chosen by researchers and practitioners [1], [4].
Lottery Ticket Hypothesis from [2] formulates a hypothesis that for every non-pruned neural network, there exists a smaller subnetwork that matches or exceeds results of the original. The algorithm originally used to obtain examples of such networks is iterative magnitude pruning with weight rewinding, and it is one of the three methods of retraining after pruning compared in this work.

2.1 Structured and Unstructured Pruning

One of the first papers about neural network pruning [6] focused solely on unstructured pruning. However, current hardware limitations do not allow to take full advantage of this form of pruning. Structured pruning is a workaround to this problem. In structured pruning, we remove the basic building blocks of a network instead of single connections. In the case of dense linear neural networks, these structures are neurons and their connections – neuron's inputs and outputs. Depending on the network's type, this can be something else. In every case, it should be a minimal unit such that the remaining neural network can be represented as a smaller, but still dense (non-pruned) neural network. In the case of structured pruning of convolutional neural networks, whole channels and their corresponding parameters in convolutional filters are removed.

3 Scope of reproducibility

Our reproducibility study tries to confirm claims from [1]. Following claims were formulated:

Claim 1: Widely used method of training after pruning: finetuning yields worse results than rewinding based methods (supported by figures 2, 3, 1, 4 and Table 5)

Claim 2: Newly introduced learning rate rewinding works as good or better as weight rewinding in all scenarios (supported by figures 2, 3, 1, 4 and Table 5, but not supported by Figure 5)

Claim 3: Iterative pruning with learning rate rewinding matches state-of-the-art pruning methods
(supported by figures 2, 3, 1, 4 and Table 5, but not supported by Figure 5)

4 Methodology

We aimed to compare three retraining approaches: 1) finetuning, 2) weight rewinding and 3) learning rate rewinding. Our general strategy that repeated across all experiments was as follows:

1. train a neural network to convergence,

2. prune the network using magnitude criterion: remove parameters with the smallest absolute value,
3. retrain the network using one of the three retraining approaches.

In the case of structured pruning: in step 2, we removed structures (either neurons or convolutional channels) with the smallest L1 norm [7], rather than removing separate connections.
In the case of iterative pruning: the network in step 1 was not randomly initialized, but instead: weights from a model from a previous iterative pruning step were loaded as the starting point. Then the three steps were repeated. On the other hand, one-shot pruning is a procedure where pruning is done only once, so there was only one cycle. In some methods, this might be done on a randomly initialized neural network, like in [5]. Here, however, one-shot pruning is done after the network reaches convergence. So the three steps are not repeated in case of one-shot pruning.
We trained all our networks using Stochastic Gradient Descent with Nesterov Momentum [8]. The learning rate was decreased in a piecewise manner during the training, but momentum coefficient was constant and equal to 0.9.

5 Model descriptions

In this report, we were focusing on an image recognition task using convolutional neural networks [9]. For most of our experiments, we chose to use identical architectures as [1] to better validate their claims and double-check their results, rather than only provide additional ones. Therefore, most of the used networks are residual networks, which were originally proposed in [10]. Additionally, to verify the general usefulness of pruning and retraining methods proposed in [1] we extend the list of tested network architectures to much larger wide residual networks from [3].

5.1 Residual networks (ResNet)

Just as [1], we chose to use the original version of ResNet as described in [10] rather than the more widely used, improved version (with preactivated blocks) from [11]. We created the models ourselves, using TensorFlow [12] and Keras. We strove to replicate the exact architectures used by [1] and [10] and train them from scratch.

Model	Trainable parameters	Kernel parameters	CIFAR-10	CIFAR-100
ResNet-20	272 282	270 896	92.46%	–
ResNet-56	855 578	851 504	93.71%	71.90%
ResNet-110	1 730 522	1 722 416	94.29%	72.21%

Table 1. ResNets architecture summary, including baseline accuracy across datasets.

ResNet hyperparameters — Learning rate started with 0.1 and was multiplied by 0.1 twice, after 36 000 and 54 000 iterations. One training cycle had 72 000 iterations in total. For all batch normalization layers, we set the batch norm decay to 0.997, following [1], which is also the default used in the original TensorFlow implementation[1]. We initialize network's weights with what is known as He uniform initialization from [13]. We regularize

[1] https://github.com/tensorflow/models/blob/r1.13.0/official/resnet/resnet_model.py

ResNets, during both training and finetuning, using $L2$ penalty with 10^{-4} coefficient. In other words, the loss function (from which we calculate the gradients) looks as follows:

$$L = \mathrm{CC}(y, p) + 10^{-4} \times \sum_{i \in W} w_i^2 \tag{1}$$

where:

L = value of the final loss function

CC = categorical crossentropy loss function

y = ground truth label of a sample or batch

p = model's prediction

W = parameters of the model

5.2 Wide Residual Networks (Wide ResNet, WRN)

WRN networks were introduced in [3]. They are residual networks created by simply increasing the number of filters in preactivated ResNet networks [11].

Model	Trainable parameters	Kernel parameters	CIFAR-10
WRN-16-8	10 961 370	10 954 160	95.72%

Table 2. Wide ResNet architecture summary.

WRN hyperparameters — As Wide ResNets are newer and much larger than ResNets, hyperparameters are slightly different. To choose them, we follow [3]. Learning rate starts with 0.1 and multiplied by 0.2 thrice: after 32 000, 48 000 and 64 000 iterations. Training lasts for 80 000 iterations. For all batch normalization layers, we use hyper-parameters from the newer TensorFlow implementation[2] with batch norm decay set to 0.9. Following [3], we use larger $L2$ penalty for this network: 2×10^{-4}. Finally, the loss function is as follows:

$$L = \mathrm{CC}(y, p) + 2 \times 10^{-4} \times \sum_{i \in W} w_i^2 \tag{2}$$

where:

L = value of the final loss function

CC = categorical crossentropy loss function

y = ground truth label of a sample or batch

p = model's prediction

W = parameters of the model

[2] https://github.com/tensorflow/models/blob/r2.5.0/official/vision/image_classification/resnet/resnet_model.py

5.3 Datasets

CIFAR-10 and CIFAR-100 are image classification datasets introduced in [14]. Following [1], we use all (50 000) training examples to train the model.

Dataset	Training examples	Validation examples	Classes	Resolution
CIFAR-10	50 000	10 000	10	32×32
CIFAR-100	50 000	10 000	100	32×32

Table 3. CIFAR datasets description.

5.4 Preprocessing and data augmentation

We used a standard data processing for both CIFAR-10 and CIFAR-100 datasets [1], [2], [3]. During training and just before passing data to the model, we:

1. standardized the input by subtracting the mean and dividing by the std of RGB channels (calculated on training dataset),
2. randomly flipped in horizontal axis,
3. added a four pixel reflection padding,
4. randomly cropped the image to its original resolution.

During the validation, we did only the first step of the above.

5.5 Experimental setup and code

Our ready-to-use code, which includes experiment definitions, can be found at https://github.com/gahaalt/reproducing-comparing-rewinding-and-finetuning. It's written using TensorFlow [12] version 2.4.2 in Python. More details are included in the repository.

5.6 Computational requirements

Recreating the experiments required a modern GPU, training all models on CPU was virtually impossible. Training time varies depending on a lot of factors: network variation and size, exact version of the deep learning library, and even the operating system. In our case, using TensorFlow 2.4.2 on Ubuntu and a single RTX 3080 GPU, the smallest of the used models, ResNet-20, takes about 20 minutes to train on CIFAR-10 dataset. To replicate our experiments, training at least a single baseline network and then, once more, a single pruned network, is required. To reduce computational requirements, we reused one non-pruned baseline for multiple compression ratios. Approximated training time requirements can be seen in the table below.

Model	Dataset	Number of updates	Updates per second	Training time
ResNet-20	CIFAR-10	72 000	59.0	22 min
ResNet-56	CIFAR-10	72 000	28.6	43 min
ResNet-110	CIFAR-10	72 000	15.9	77 min
WRN-16-8	CIFAR-10	80 000	17.4	78 min

Table 4. Time requirements for replicating or running experiments from this report. Reported times are obtained using a single RTX 3080 GPU in Linux environment, using TensorFlow in version 2.4.2.

For all our experiments together, we estimate the total number of GPU hours spent to be around 540.

6 Method description

We compare three methods of retraining after pruning. For all of them, the starting point is a network that was already trained to convergence, then pruned to a desired sparsity. The difference between the three retraining methods is what follows after it.

6.1 Fine-tuning

Fine-tuning is retraining with a small, constant learning rate – in our case, whenever fine-tuning was used, the learning rate was set to 0.001 as in [1]. We finetune the network for the same number of iterations as the baseline – 72 000 iterations in the case of the original ResNet architecture. In this method, such long retraining would not be necessary in practical applications, since the network converges much faster.

6.2 Weight rewinding

Weight rewinding restores the network's weights from a previous point (possibly beginning) in the training history and then continues training from this point using the original training schedule – in our case a piecewise constant decaying learning rate schedule. When rewinding a network to iteration K that originally trained for N iterations: first prune the non-pruned network that was trained for N iterations. Then, for connections that survived, restore their values to K-th iteration from the training history. Then train to the convergence for the remaining $N - K$ iterations.

6.3 Learning rate rewinding

Learning rate rewinding continues training with weights that have already converged, but restores the learning rate schedule to the beginning, just as if we were training from scratch, and then trains to the convergence once again. This reminds the cyclical learning rates from [15]. Learning rate rewinding really is weight rewinding for $K = N$, but the final retraining is always for N iterations.

7 Results reproducing original paper

In most of our experiment, just as [1], we investigate how does the trade-off between prediction accuracy and compression ratio look like. In one of the experiments (Table 5)

we verify only one compression ratio, but for the rest, we verify multiple. We report a median result out of 2 up to 12 trials for each compression ratio. To better utilize our compute capabilities, we decided to spend more training cycles in situations where there is no clear winner between the compared methods. On each plot, we include error bars showing 80% confidence intervals.
In this section, we include experiments that we successfully reproduced. Most of them match the original ones within 1% error margin. We noticed some of our results were slightly better than authors of [1] originally reported.
Across all scenarios where finetuning was tested, it was by far the worst of the three methods, which directly supports claim 1 (Section 3). Weight rewinding and learning rate rewinding most often are equally matched, but in some cases learning rate rewinding works a little better.

7.1 ResNets on CIFAR-10 dataset

Results we observe here are consistent with what we see in [1], [2]. Iterative pruning is better than one-shot pruning, but more time consuming. In extreme cases, iterative pruning requires 20 times as many iterations than one-shot pruning to complete. But it is not as bad for moderate sparsity pruning. Larger networks work better than smaller ones. Even when the number of parameters left after pruning is the same – originally larger network will outperform the smaller one.
Out of the retraining methods, weight rewinding and learning rate rewinding seem to be similar, but finetuning is visibly worse. In some cases, learning rate rewinding outperforms weight rewinding. Similar conclusions can be drawn from both structured and unstructured pruning results.

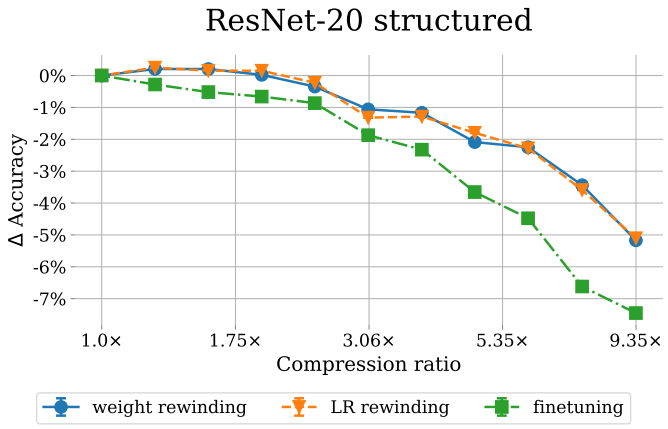

Figure 1. Results of ResNet-20 (Table 1) on CIFAR-10 (Table 3) with structured, one-shot, magnitude pruning. Results show varying compression ratios. Maximal compression ratio (9.35×) means that there are only 29 000 non-zero kernel parameters left in ResNet-20.

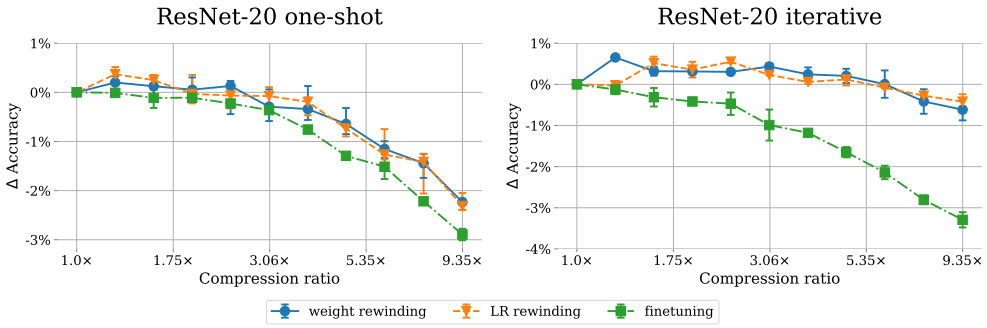

Figure 2. Results of ResNet-20 (Table 1) on CIFAR-10 (Table 3) with unstructured, magnitude pruning in versions: one-shot and iterative. Results show varying compression ratios. Maximal compression ratio (9.35×) means that there are only 29 000 non-zero kernel parameters left. This experiment supports claims 1, 2, 3 (Section 3).

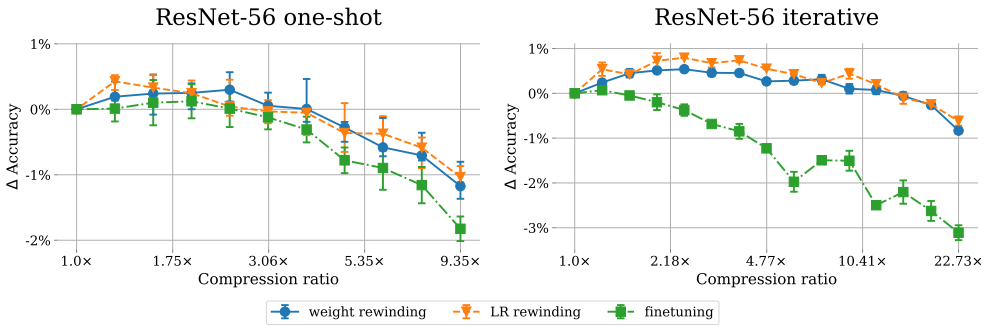

Figure 3. Results of ResNet-56 (Table 1) on CIFAR-10 (Table 3) with unstructured, magnitude pruning in versions: one-shot and iterative. Results with varying compression ratios. Maximal compression ratio means (22.73×) that there are only 37 600 non-zero kernel parameters left. This experiment supports claims 1, 2, 3 (Section 3).

Network	Dataset	Retraining	Sparsity	Test Accuracy
ResNet-110	CIFAR-10	None	0%	94.29%
ResNet-110	CIFAR-10	LR rewinding	89.3%	93.74%
ResNet-110	CIFAR-10	weight rewinding	89.3%	93.73%
ResNet-110	CIFAR-10	finetuning	89.3%	93.32%

Table 5. Results of ResNet-110 (Table 1) trained on CIFAR-10 (Table 3) with unstructured, one-shot magnitude pruning. Sparsity 89.3% corresponds to 9.35× compression ratio. This experiment supports claims 1, 2, 3 (Section 3).

8 Results beyond original paper

8.1 ResNets on CIFAR-100 dataset

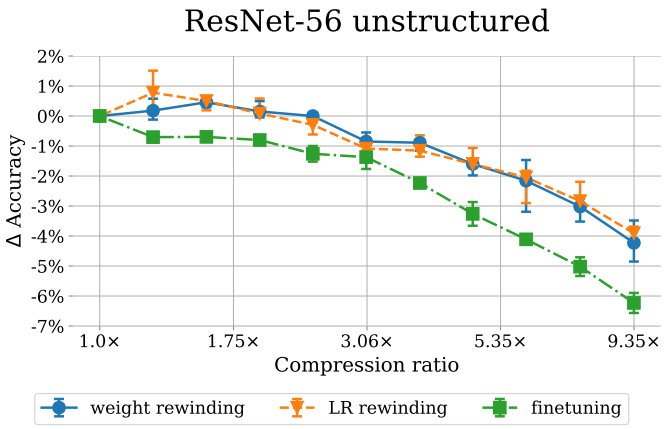

Figure 4. Results of ResNet-56 (Table 1) on CIFAR-100 (Table 3) with unstructured, one-shot, magnitude pruning. Results with varying compression ratios. Maximal compression ratio (9.35×) means that there are only 91 500 non-zero kernel parameters left. This experiment supports claims 1, 2, 3 (Section 3) even though this scenario wasn't originally tested in [1].

8.2 WRN-16-8 on CIFAR-10 dataset

WRN-16-8 shows consistent behaviour – accuracy in the low sparsity regime is reduced in comparison to the baseline. In the case of iterative pruning, where each step is another pruning in the low sparsity regime, it leads to a large difference between the two retraining methods. Since for WRN-16-8 one-shot, low sparsity pruning shows a small accuracy regression in comparison to the baseline, this regression accumulates when pruning multiple times, which we do in iterative pruning. We think that the degradation we observe in the case of iterative pruning is an effect of stacking multiple smaller defects that we observe in the case of one-shot pruning. This can be seen in Figure 5.

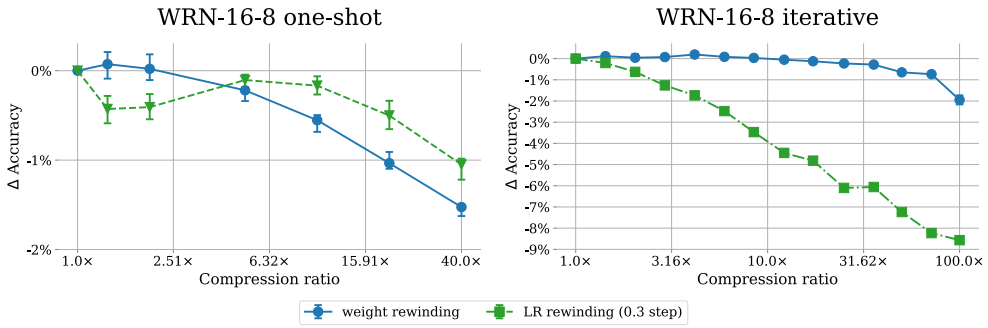

Figure 5. Results of WRN-16-8 (Table 2) on CIFAR-10 (Table 3) with unstructured, magnitude pruning in versions: one-shot and iterative. Results with varying compression ratios. Maximal compression ratio (100×) leaves 109 500 non-zero kernel parameters while achieving around 94% accuracy or around 95% when leaving 153 400 non-zero parameters. One can see catastrophic effects of high-sparsity pruning when using learning rate rewinding procedure.

For iterative pruning (figures 2, 3, 5) we used a nonstandard step size of 30% per iterative pruning iteration, which was a way to reduce the computational requirements. We believe this choice does not change the validity of the claims, and to support it, we provide a comparison of our step size to the more commonly used 20%. We show that there is virtually no difference between both versions and the aforementioned catastrophic degradation occurs in both cases, as long as the step size is in the low sparsity regime.

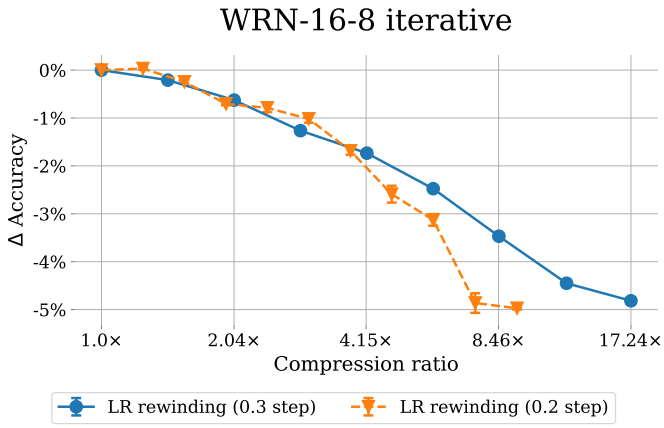

Figure 6. Results of WRN-16-8 (Table 2) on CIFAR-10 (Table 3) with unstructured, iterative, magnitude pruning with two different step sizes. Results show varying compression ratios and accuracy.

9 Discussion

We were able to confirm the general conclusion of [1]. Fine-tuning can be mostly replaced by other retraining techniques, e.g., by weight rewinding as it was done by [2]. We confirmed that learning rate rewinding worked even better in some scenarios. However, we have also shown in Figure 5 that the newly proposed learning rate rewinding is a poor choice when we are pruning large networks – in our case that is WRN-16-8. We believe this should be examined further as there might exist a simple workaround to this problem – a retraining procedure between weight rewinding and learning rate rewinding, which works even for larger networks. Furthermore, it would be interesting to see what happens in the network when this catastrophic accuracy degradation occurs. Perhaps, the reason for it not occurring with the original ResNet, but occurring with larger architectures, is the degree to which the larger networks overtrain – larger networks tend to overfit more. And such an overfitted network might be not a good starting point for the retraining.

Acknowledgements

The authors thank Polish National Science Center for funding under the OPUS-18 2019/35/B/ST6/04379 grant and the PlGrid consortium for computational resources.

References

1. A. Renda, J. Frankle, and M. Carbin. “Comparing Rewinding and Fine-tuning in Neural Network Pruning.” In: (2020). arXiv: 2003.02389. URL: http://arxiv.org/abs/2003.02389 (visited on 06/02/2020).

2. J. Frankle and M. Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks." In: *7th International Conference on Learning Representations, ICLR 2019* (2019). arXiv: 1803.03635 Citation Key: Frankle2019, pp. 1–42.
3. S. Zagoruyko and N. Komodakis. "Wide Residual Networks." In: *Proceedings of the British Machine Vision Conference (BMVC)*. Ed. by E. R. H. Richard C. Wilson and W. A. P. Smith. BMVA Press, Sept. 2016, pp. 87.1–87.12. DOI: 10.5244/C.30.87. URL: https://dx.doi.org/10.5244/C.30.87.
4. Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. "Rethinking the value of network pruning." In: *7th International Conference on Learning Representations, ICLR 2019* (2019). arXiv: 1810.05270 Citation Key: Liu2019a, pp. 1–21.
5. N. Lee, T. Ajanthan, and P. H. S. Torr. "SNIP: Single-shot Network Pruning based on Connection Sensitivity." In: *CoRR* abs/1810.02340 (2018). arXiv:1810.02340. URL: http://arxiv.org/abs/1810.02340.
6. Y. LeCun, J. Denker, and S. Solla. "Optimal Brain Damage." In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1990. URL: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.
7. E. J. Crowley, J. Turner, A. Storkey, and M. O'Boyle. "A Closer Look at Structured Pruning for Neural Network Compression." In: 10 (2018). arXiv: 1810.04622 Citation Key: Crowley2018, pp. 1–12. URL: http://arxiv.org/abs/1810.04622.
8. A. Botev, G. Lever, and D. Barber. *Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent*. 2016. DOI: 10.48550/ARXIV.1607.01981. URL: https://arxiv.org/abs/1607.01981.
9. Y. LeCun. "Handwritten Digit Recognition with a Back-Propagation Network." In: *Neural Information Processing Systems*. American Institute of Physics, 1988. URL: https://proceedings.neurips.cc/paper/1987/file/a684eceee76fc522773286a895bc8436-Paper.pdf (visited on 07/14/2021).
10. K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem (2016). arXiv: 1512.03385 Citation Key: He2016 ISBN: 9781467388504, pp. 770–778. DOI: 10.1109/CVPR.2016.90.
11. K. He, X. Zhang, S. Ren, and J. Sun. "Identity mappings in deep residual networks." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9908 LNCS (2016). arXiv: 1603.05027 ISBN: 9783319464923, pp. 630–645. DOI: 10.1007/978-3-319-46493-0_38.
12. Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.
13. K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.
14. A. Krizhevsky, V. Nair, and G. Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)." In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.
15. L. N. Smith. *Cyclical Learning Rates for Training Neural Networks*. 2017. arXiv:1506.01186 [cs.CV].