Post-training deep neural network pruning via layer-wise calibration

Ivan Lazarevich
Intel Corporation
ivan.lazarevich@intel.com

Alexander Kozlov
Intel Corporation
alexander.kozlov@intel.com

Nikita Malinin
Intel Corporation
nikita.malinin@intel.com

Abstract

We present a post-training weight pruning method for deep neural networks that achieves accuracy levels tolerable for the production setting and that is sufficiently fast to be run on commodity hardware such as desktop CPUs or edge devices. We propose a data-free extension of the approach for computer vision models based on automatically-generated synthetic fractal images. We obtain state-of-the-art results for data-free neural network pruning, with ∼1.5% top@1 accuracy drop for a ResNet50 on ImageNet at 50% sparsity rate. When using real data, we are able to get a ResNet50 model on ImageNet with 65% sparsity rate in 8-bit precision in a post-training setting with a ∼1% top@1 accuracy drop. We release the code as a part of the OpenVINO™ Post-Training Optimization tool[1].

1. Introduction

Deep neural network (DNN) models have achieved unprecedented accuracy in several crucial domains such as computer vision and natural language processing. Despite the success of DNN models, an unreasonably large amount of computations and memory required for their inference limits their deployment on edge devices, such as smart cameras equipped with low-power CPUs, GPUs or ASIC accelerators. Significant efforts in recent years have been devoted to both hardware design and algorithmic approaches to DNN model compression to enable inference speedups for various model architectures and use cases. Some of the DNN compression methods, such as 8-bit quantization, were adapted to the post-training setting where the original DNN model to be compressed could come from any software framework and no access to the original training pipeline and the training dataset is given. One of the promising approaches to reduce the memory footprint and inference latency of DNNs is weight pruning [2, 4], which results in models with sparse weight matrices. Recently, a lot of research and development has been aimed at leveraging weight sparsity to achieve inference speedups on a range of hardware platforms [3, 6]. However, relatively little effort was devoted to providing accurate sparse DNN models in the post-training scenario.

In this work, we propose a recipe for fast post-training pruning of DNNs that produces models with significant sparsity rates (e.g. 50%) but negligible accuracy drops. Furthermore, if combined with weight quantization techniques, the proposed method could reduce the model memory footprint by a factor of 6-8x [16]. We propose a fast data-free extension of our weight pruning pipeline which allows getting state-of-the-art accuracy levels for a range of computer vision models. To streamline the deployment process of sparse quantized DNNs on hardware, we have implemented the proposed method as a part of the OpenVINO™ Post-Training Optimization tool.

We summarize our contributions as follows:

- A recipe for post-training weight pruning with demonstrated results on a wide range of models and datasets.
- State-of-the-art results for data-free weight pruning of computer vision models using synthetic fractal images for model compression.
- An ablation study of the proposed post-training weight pruning pipeline demonstrating the effects of particular components such as per-layer sparsity rate selection criteria, bias correction and layer-wise fine-tuning settings.

2. Related work

Neural network weight pruning is a technique used to produce lightweight models by removing (zeroing out) a certain percentage of unimportant weights. In this work, we focus on unstructured pruning (weight sparsification) whereby no structural constraints on the sparsity pattern are imposed and a subset of weights determined to have the lowest importance score values is removed regardless of position in weight tensors. Various definitions of weight

[1] https://docs.openvinotoolkit.org/latest/pot_README.html

Table 1. Accuracy values of the sparse 8-bit quantized DNN models obtained with the proposed post-training method. Metric values were measured on a CPU. The same is for other accuracy values reported in the paper unless specified otherwise.

Model	Dataset (acc. metric)	Sparsity rate, %	Compressed model acc.	Absolute acc. drop
ResNet50	ImageNet (top@1 acc.)	65	75.09	1.04
ResNet18	ImageNet (top@1 acc.)	50	68.93	0.81
GoogleNetV4	ImageNet (top@1 acc.)	50	78.96	0.94
MobileNetV2	ImageNet (top@1 acc.)	40	70.29	1.51
MobileNetV1-SSD	VOC07 (mAP)	50	71.53	0.98
TinyYOLOv2	COCO (AP)	50	28.29	0.83
NCF	MovieLens 20M (hit ratio)	70	64.67	0.93
BERT-base	MRPC (acc.)	50	82.50	0.63

Table 2. Accuracy of sparse computer vision models obtained with layer-wise fine-tuning on different input data. Note that accuracy levels are very similar when fine-tuning on original validation or training datasets, suggesting the absence of overfitting during layer-wise fine-tuning. ”FractalDB-1k(c)” denotes the colored FractalDB-1k dataset.

Model (sparsity rate, dataset/acc. metric)	Orig. model acc.	Val. data	Training data	FractalDB 1k(c)	White noise
ResNet18 (50%; ImageNet top@1)	69.75	68.94	68.92	68.27	66.90
ResNet50 (50%; ImageNet top@1)	76.13	75.51	75.57	74.50	73.89
MobileNetV2 (40%; ImageNet top@1)	71.81	70.04	70.12	68.94	66.84
MobileNetV1-SSD (50%; VOC07 mAP)	72.51	71.37	71.53	71.13	69.52
TinyYOLOv2 (50%; COCO AP)	29.12	28.06	28.29	28.18	27.10
RetinaFace-ResNet50 (50%; WIDER FACE mAP)	87.29	87.40	87.41	87.45	86.87

importance functions have been proposed in the literature [4, 23], the simplest baseline being magnitude-based weight pruning, as well as various heuristics to determine per-layer sparsity rates [13, 7]. Magnitude-based sparsification via a global importance threshold was found to be a strong baseline in the compression-aware training regime for a range of models [4, 20]. These weight pruning approaches typically imply compression-aware model re-training, which means existing access to the training code, the training dataset and appropriate compute resources. Other DNN compression techniques, such as 8-bit quantization, however, have been successfully applied in a less restrictive setting – in the post-training or data-free regimes [18]. The post-training compression regime is favorable from a practical perspective, since model compression could be ultimately implemented via a single API call rather than via the modification of the original model training code. There, however, have been few attempts to implement post-training or data-free weight pruning of DNNs, primarily due to the large accuracy drop incurred during sparsification [8, 21]. Recently, there have also been developed layer-wise gradient optimization-based methods for post-training compression [9, 15, 17, 8] with applications to low-bitwidth quantization and weight pruning. These methods are promising because they allow restoring compressed model accuracy in the post-training setting in many cases. Nevertheless accuracy degradation was still found to be significant for sparsity rates above 40%. In this work, we propose a post-training sparsification recipe that allows insignificant accuracy drops on a range of DNN models at sparsity rates of 50% and higher. We also suggest a straightforward and fast extension of the method for the data-free compression of computer vision models, using synthetic fractal image data, that allows getting state-of-the-art accuracy on a range of natural image datasets.

3. Post-training sparsity pipeline

The proposed post-training sparsity pipeline consists of three basic steps: (i) layer-wise sparsity rate selection given a global sparsity constraint, (ii) bias & variance correction steps, and (iii) layer-wise fine-tuning using auxiliary knowledge distillation losses. We introduce a progressively increasing sparsity schedule for each layer whereby these three steps are performed iteratively and the global sparsity rate in the model is increased on each iteration (see the flowchart in Fig. 1). The global sparsity rate for the model on the t^{th} iteration of the pipeline is determined via the following polynomial (cubic) sparsity schedule [23]:

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t}{T}\right)^3 \tag{1}$$

where s_i and s_f are the initial and final global sparsity rates of the model, respectively, and T is the total number of iterations of the pipeline. After the original floating-point precision model with the target global sparsity rate is ob-

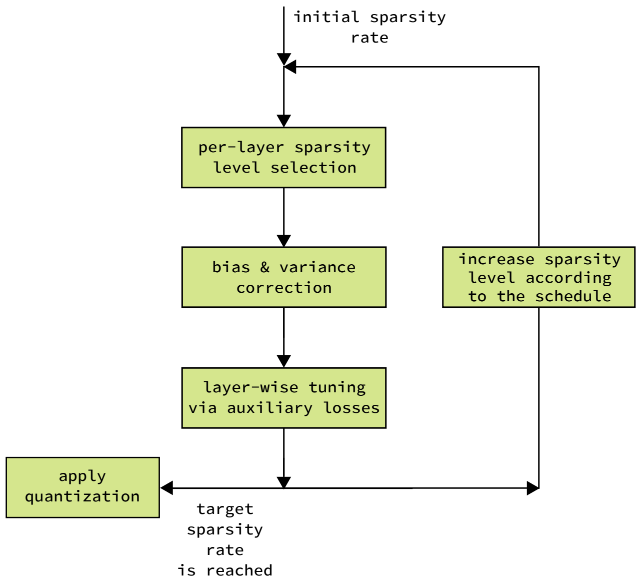

initial sparsity rate
per-layer sparsity level selection
bias & variance correction
layer-wise tuning via auxiliary losses
increase sparsity level according to the schedule
apply quantization
target sparsity rate is reached

Figure 1. Flowchart of the proposed post-training sparsity pipeline. The process begins with a small initial global sparsity value, the model is fine-tuned in a layer-wise manner, the sparsity level is increased and the fine-tuning is repeated. This iterative process is carried out until the target sparsity level or the maximal allowed accuracy drop is reached (either of these parameters is set in advance).

tained, the standard procedure of post-training quantization is performed to prepare the model to be executed in 8-bit precision. We found that performing post-training quantization on the pruned model does not incur significant accuracy degradation compared to the original-precision sparse model (see Fig. 2 for results on ResNet18/50 on ImageNet), probably due to reduced quantization noise of sparse weight matrices. We are using the following quantization configuration throughout the paper: symmetric per-tensor quantization of activations (except for specific per-channel cases like e.g. depthwise convolutions) and symmetric per-channel quantization of weights. We further provide details on all the steps performed on every iteration of the pruning pipeline in the corresponding sections below.

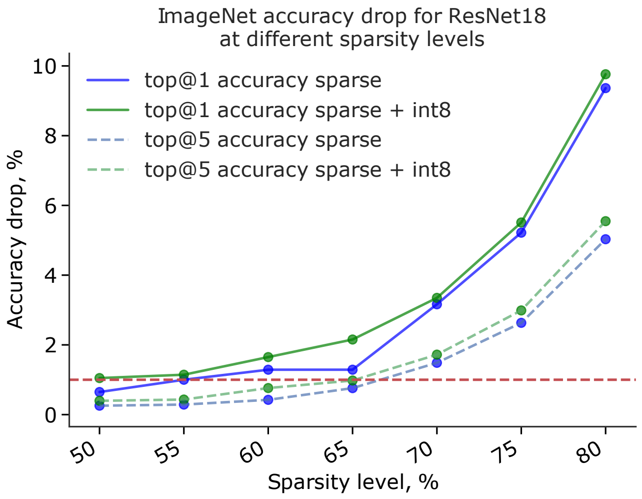
3.1. Layer-wise sparsity rate selection procedure

The problem of selecting an optimal (in terms of model accuracy) layer-wise sparsity rate configuration given a certain global sparsity constraint is a widely discussed problem in the literature [13]. The proposed approaches range from simple heuristics (e.g. pruning uniformly except for the first and the last layers in the network [4]) to making per-layer sparsity rates learnable [12] or searching for the best configuration via global non-gradient optimization or reinforcement learning [7]. The heuristic approaches also include finding a global threshold for weight importance scores and pruning all the weights with importances below

Table 3. Impact of per-layer sparsity selection criteria on a pre-trained ResNet50, ResNet18 and MobileNetV2 models. ResNets are pruned at a 50% sparsity rate, while the sparsity rate for MobileNetV2 is 30% and bias correction is applied to MobileNetV2 (see main text for details).

Sparsity selection criterion	BN-fusing	Top@1 accuracy, %
Original model (ResNet50)		76.13
Magnitude	Yes	0.3814
L2-normalized magnitude	Yes	**72.544**
LAMP	Yes	72.328
Magnitude	No	**72.831**
L2-normalized magnitude	No	72.244
Original model (ResNet18)		69.75
Magnitude	Yes	0.418
L2-normalized magnitude	Yes	64.856
LAMP	Yes	**64.866**
Original model (MobileNetV2)		71.81
Magnitude	Yes	18.386
L2-normalized magnitude	Yes	**69.528**
LAMP	Yes	69.524

this threshold. This naturally leads to a non-uniform pattern of per-layer sparsity rates. The importance score function in this case might be the absolute weight magnitude or a normalized version thereof (like e.g. the LAMP score [13]). We compared several variations of importance score functions in the global threshold approach, namely (i) the absolute weight magnitude, (ii) the absolute weight magnitude normalized by the L2 norm of the corresponding layer, (iii) the LAMP score (Table 3). We initally observed that the global magnitude criterion led to much worse accuracy comprared to the normalized criteria (Table 3). This effect was found to be caused by the fusing of the BatchNorm layers into preceding convolutions, which was performed in the model prior to compression. BatchNorm fusing resulted in different layer-wise weight scales compared to the original model, an effect easily counteracted by per-layer normalization of weight magnitudes. In the case where the normalization layers were not fused into convolutions, however, we found that the vanilla global magnitude criterion performed the best compared to LAMP and L2-normalized magnitude (Table 3 and Figure 3). We further assumed that the fusing could generally occur prior to model compression and the original normalization layer parameters might



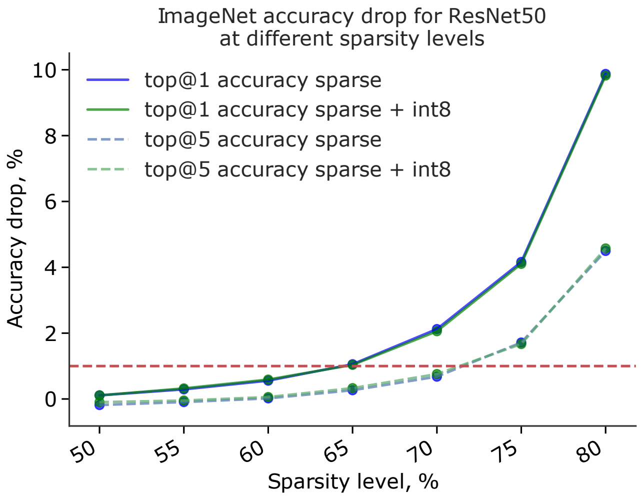


Figure 2. Accuracy drop/sparsity rate curves for a ResNet18 (top) and a ResNet50 (bottom) model obtained with our post-training pruning and quantization pipeline. The horizontal dashed red line indicates the level of 1% absolute accuracy drop. Note that post-training quantization of the pruned model does not lead to a huge accuracy drop increase for both models at different sparsity rate levels.

be unknown, hence we picked the per-layer L2-normalized magnitude criterion as our sparsity rate selection heuristic. It performed better than LAMP in our post-training scenario on most of the models with BatchNorm fusing. The weight importance criterion for the i^{th} weight in the l^{th} layer w_i^l we use in our pipeline thus reads

$$I(w_i^l) = \frac{|w_i^l|}{\sqrt{\sum_{j \in l} |w_j^l|^2}} \tag{2}$$

We pool the importance scores from all the layers and find the threshold value corresponding to the set sparsity rate. The weights with importance values below the threshold are pruned.

3.2. Weight and activation bias correction

Once the layer-wise pruning rates have been determined, the weights are zeroed out based on the intra-layer absolute magnitudes. This pruning operation distorts the weight distribution, introducing bias and scale shifts. It is beneficial to carry out a bias correction procedure on the weights in order to restore the original mean and variance values in all of the convolutional layer filters and fully-connected layer weight matrices [1]. We perform the following affine transformation on all of the pruned weight tensors in a per-channel/per-feature fashion:

$$W^s_{corr} = \lambda W^s + E(W_{dense}) - E(\lambda W^s) \tag{3}$$

$$\lambda = \frac{\sigma(W_{dense})}{\sigma(W^s) + \epsilon} \tag{4}$$

where W^s_{corr} is the weight tensor after the correction procedure, and W_s and W_{dense} are the weight tensors of sparse and original dense models, respectively, and E and σ are the mean and standard deviation operators, $\epsilon = 10^{-9}$ is a small constant added for numerical stability. The resulting sparse weight tensor has the same mean and variance values as original dense model weights for each output kernel/feature, since this correction is applied to every output feature independently. Output activations at each pruned layer are also suffering from a bias introduced by the zeroed weights, which can be compensated by altering the bias parameters of the convolutional and fully-connected layers. Nagel et al. [18] proposed to perform this operation to mitigate biases introduced by quantization in an iterative fashion, correcting the first layer and then calculating the bias shift factors for the second layer using this corrected model. We found that a one-shot version of the bias correction procedure was sufficient for post-training sparsity, whereby we perform a forward pass of the original model and calculate the input activation tensors X_{dense} for each layer. The corrected bias parameters are then determined as

$$b_{corr} = b_{dense} + E(f(W_{dense}, X_{dense})) - E(f(W^s_{corr}, X_{dense})) \tag{5}$$

where $f(W, X)$ is the convolutional or matrix-multiply operation of the layer acting on inputs X with weights W, b_{dense} are the original bias values in the layer, X_{dense} is the set of input activation tensors for the corresponding layer in the original dense model. In other words, we are using the input tensors from the original model to calculate bias shifts, not from the iteratively corrected compressed model. We found no significant difference in the resulting accuracy between the two approaches, with the one-shot one being faster since it requires a single forward pass of the

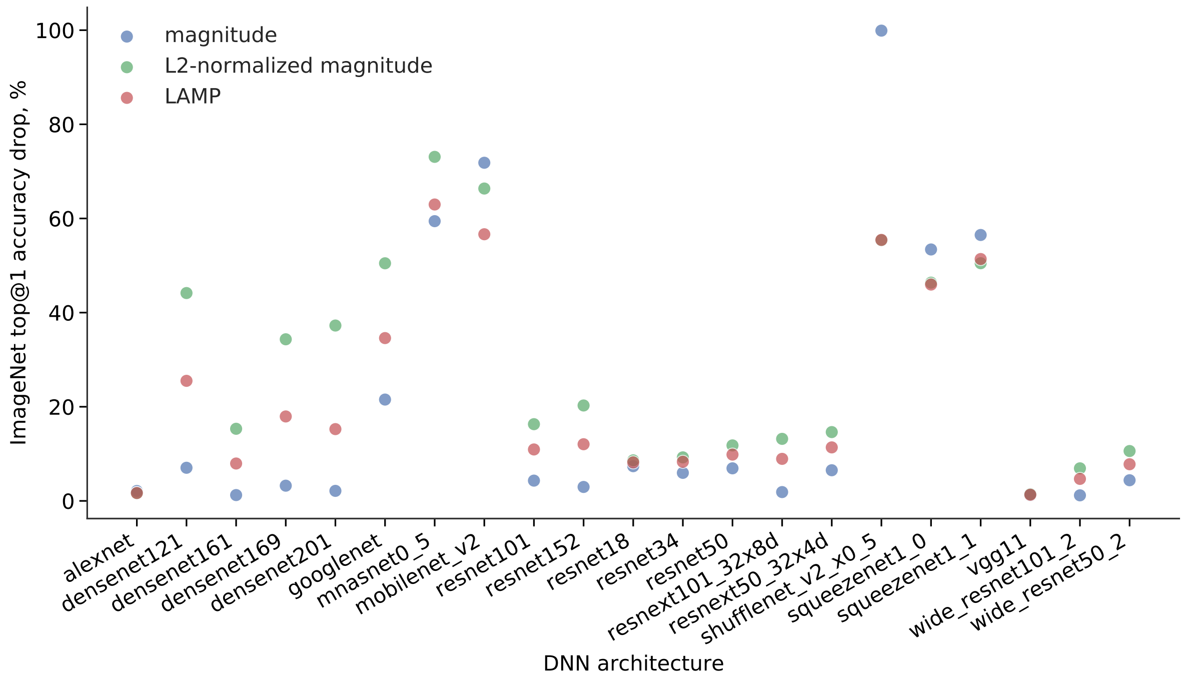


Figure 3. Absolute top@1 accuracy drops for a range of models from the torchvision package pruned with a 50% global sparsity rate depending on the per-layer compression level selection criterion. BatchNorm fusing was not performed in the networks. Pruning is done in the post-training regime (without any fine-tuning), BatchNorm adaptation is performed after weight pruning. Global magnitude criterion is optimal in most cases in this setting except for lightweight models such as MobileNets, SqueezeNets and ShuffleNets. BatchNorm adaptation is a procedure analogous to bias correction whereby the BN statistics are recollected after the model has been compressed [14]. Accuracy drops are measured relative to the original pre-trained weights using PyTorch on a GPU.

model. Results of the weight & activation bias correction procedures are shown in Table 4 for a ResNet18 model at 50% sparsity rate. Both procedures cumulatively improve the pruned model accuracy and top@1 accuracy drops are not exceeding several percent for many ImageNet models at the sparsity rate of 50% just after layer-wise sparsity selection and bias correction. Accuracy can be further improved by local layer-wise fine-tuning using auxiliary knowledge-distillation losses, which is described in more detail below.

3.3. Local layer-wise fine-tuning with auxiliary losses

Gradient-based fine-tuning with auxiliary loss functions was previously successfully applied for post-training quantization [17, 15, 9] and, to a smaller extent, to post-training weight pruning [8] and even filter pruning in convolutional networks [5]. In this work, we follow a similar approach whereby we define a local knowledge distillation loss for every pruned layer (see Figure 4). These losses serve as a measure of how close the output activation feature maps of the original (unpruned) and pruned layers are. Suppose the pruned layer weights and biases are W^s and b^s, then the knowledge distillation mean-squared error loss for that layer is defined as

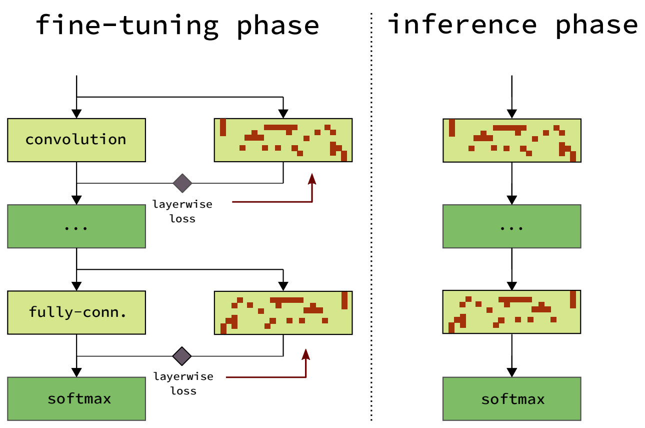


Figure 4. Schematic description of the layerwise fine-tuning approach for post-training sparse model calibration. Input and dense model output tensors are pre-computed and stored in memory for each tuned layer. The red arrows depict the local flow of gradients during weight and bias optimization. Red pixels indicate sparsity masks in the compressed layers (sparsity levels are individually selected for each layer).

$$L = \sum_{i \in batch} (Y^i_{dense} - f(W^s M^s, X^i_{dense}) - b^s)^2$$

$$Y^i_{dense} = f(W_{dense}, X^i_{dense}) \quad (6)$$

where $f(X, W)$ is the convolutional/matrix-multiply operation represented by the layer with weights W acting on inputs X. The input tensors X_{dense} used to calculate the output activations above are constructed by running a forward pass of the original, unpruned model. M^s is the binary mask layer which is equal to one if the corresponding weight is not pruned and zero otherwise. We fix the sparse binary mask and run a gradient descent of the loss functions defined above for each layer independently to find the optimal weights and biases W^s, b^s.

3.4. Ablation study

3.4.1 Layer-wise fine-tuning settings.

We run several ablation experiments to establish the best optimization settings for the layer-wise fine-tuning procedure, since it is inherently different from full model training via backpropagation. We used a batch size of 50 samples in our experiments, and found the optimal learning rate values across different models to be 10^{-5} for weights and 10^{-4} for bias parameters. We found that using the Adam optimizer outperforms alternatives, such as SGD with momentum or Adadelta, and techniques for better generalization in the vicinity of local minima, like Lookahead [22] and Stochastic Weight Averaging [10] were also not found to be beneficial in the layer-wise fine-tuning case. The MSE loss function also was found to be a better choice than e.g. L1 loss, Huber loss or cosine similarity between feature maps. We did not observe significant over-fitting present in layer-wise optimization (we discuss this phenomenon in more details below) and in particular we found that even low values of weight decay/L2 weight regularization strength such as 10^{-6} could hurt the resulting model accuracy (see Table 5). *Thus, we set the weight decay strength to 0 in all our experiments.* Increasing model sparsity rate using a cubic schedule throughout the pruning pipeline also turned out to improve accuracy for most models compared to the constant sparsity baseline (Table 6). Overall, we were able to prune and quantize a wide range of models with resulting sparsity rates ranging from 40% to 70% and an absolute accuracy drop not exceeding or close to 1% with our layer-wise fine-tuning recipe using images from the respective models' training datasets (Table 1 and Figure 2).

3.4.2 Source of input data used for fine-tuning.

Throughout experiments, we noticed that the set of input samples to be used for fine-tuning does not have to be necessarily large (we used a pool of randomly selected several hundred samples in our experiments) and can come either from the training or the validation dataset, with no significant accuracy difference between the two (see Table 2). These results suggest that this fine-tuning regime is not as prone to over-fitting compared to full model training, a fact that was also previously reported for layer-wise tuning of a quantized model [9]. We observed no difference between fine-tuning on a batch of training or validation samples not only for the ImageNet dataset but also for object detection models trained on the Pascal VOC, COCO and WIDER FACE datasets. This lack of over-fitting is not surprising since no annotation is used during fine-tuning and all the layers are optimized independently, which reduces the amount of tuned parameters per single optimization problem. The amount of supervision signal is also high because the difference between whole activation tensors produced by a set of input samples is used as a loss function. We further verified whether we could utilize arbitrary input data for bias correction and layer-wise fine-tuning, not related to the original dataset that the model has been trained and tested on. The intuition behind this is that once the output

Table 4. Impact of bias & variance correction for weights and activations on a ResNet18 model on ImageNet with 50% of the weights pruned.

ResNet18 50% sparse	Top@1 accuracy	Top@5 accuracy
L2-normalized magnitude	64.856	86.126
L2-normalized magnitude (+ act. bias correction)	66.852	87.438
L2-normalized magnitude (+ act. & weight bias correction)	**67.46**	**87.794**

Table 5. Impact of L2 weight regularization on the fine-tuned model accuracy for a ResNet18 model at 50% sparsity on ImageNet. Even small weight decay values result in significant accuracy loss compared to the baseline with no regularization in place.

L2-reg. strength	Top@1 / Top@5 accuracy, %
$\lambda = 0.0$	**68.97 / 88.77**
$\lambda =$ 1e-6	67.78 / 87.96
$\lambda =$ 1e-5	38.14 / 64.49

Table 6. Impact of the sparsity schedule vs. constant sparsity throughout fine-tuning. All metrics are reported at 50% sparsity rates. The cubic schedule was initialized at 10% sparsity rate which was increased in 10 iterations. The same number of optimizer steps was used for both fine-tuning modes.

Model	Top@1 accuracy w/o schedule	Top@1 accuracy with schedule
ResNet18	68.76	**68.91**
ResNet50	75.43	**75.60**
GoogleNetV4	**79.37**	78.10

Table 7. Accuracy of models pruned in the post-training regime using different types of synthetic data: randomly-colored FractalDB1k images, (original) grayscale FractalDB-1k, and generated white noise images. Note that colorization of FractalDB images leads to increased resulting accuracy in most models.

Model (sparsity rate, dataset/acc. metric)	FractalDB-1k(c)	FractalDB-1k	White noise
ResNet18 (50%; ImageNet top@1)	**68.27**	67.94	66.90
ResNet50 (50%; ImageNet top@1)	**74.50**	74.46	73.89
MobileNetV2 (40%; ImageNet top@1)	**68.94**	68.47	66.84
MobileNetV1-SSD (50%; VOC07 mAP)	**71.13**	70.79	69.52
TinyYOLOv2 (50%; COCO AP)	28.18	**28.28**	27.10

activation feature maps produced by these arbitrary data are similar to the ones produced by running model inference on its original dataset, the layer-wise fine-tuning procedure could produce a model that is sufficiently accurate on the validation data. In particular, we tested several computer vision models trained on different datasets (ImageNet, Pascal VOC, COCO, WIDER FACE; see Table 2); the models were pruned using our post-training pipeline, but the input images used to calculate activation statistics and feature maps for fine-tuning consisted of synthetically generated white noise (each pixel value in every color channel is independently sampled from a uniform distribution from 0 to 255). We observed certain accuracy degradation when fine-tuning on white noise images compared to tuning on original data, but typically not exceeding several percent. We further tested whether these results could be improved by using synthetic images producing activation distributions closer to those generated by natural images in corresponding datasets. We took images from the FractalDB-1k dataset [11], which is comprised of automatically-generated grayscale images of fractals. These images and their generated annotation were used to pre-train strong backbones for computer vision, including Vision Transformers [11, 19]. We found that using these fractal images as input samples to computer vision models during post-training weight pruning significantly improves the resulting model accuracy compared to the white noise baseline (Table 2). We took the original 512x512 images from FractalDB-1k, randomly colored them by performing random shift-scale operations on the color channels and used the same pre-processing strategy as for the original datasets that the models were trained on. Overall, we were able to achieve an accuracy degradation of absolute 1-3% at 50% sparsity rates in the data-free pruning regime by using synthetically-generated fractal images as model inputs. The results generalized beyond ImageNet to other natural image datasets like Pascal VOC, COCO and WIDER FACE. Random colorization of FractalDB images consistently yielded better accuracy compared to using the (original) grayscale images (Table 7). The proposed data-free pruning approach leads to *better accuracy values compared to the existing state-of-the-art* [8] and is also fast and less restrictive since it does not include a resource-consuming data distillation process that relies on backpropagation through the model graph. This can be seen by comparing our results from Table 2 for models like ResNet18 and MobileNetV2 to those in Figure 3 of Horton et al. [8].

4. Conclusion

In this work, we have presented a novel post-training pruning recipe for deep neural networks that allows zeroing out a significant proportion of model weights without significant accuracy drops. We demonstrated efficiency of the proposed pipeline on ImageNet models, object detection models on Pascal VOC and COCO datasets as well as deep NLP and recommendation models. We proposed a data-free formulation of the method by using synthetic fractal images to compress computer vision models, which led to state-of-the-art results in data-free weight pruning. We demonstrated that the proposed pruning method can also be safely combined with post-training quantizaton, further increasing its applicability in production settings.

References

[1] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2018. 4

[2] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020. 1

[3] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14629–14638, 2020. 1

[4] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019. 1, 2, 3

[5] Hui Guan, Xipeng Shen, and Seung-Hwan Lim. Wootz: A compiler-based framework for fast cnn pruning via composability. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 717–730, 2019. 5

[6] Cong Guo, Bo Yang Hsueh, Jingwen Leng, Yuxian Qiu, Yue Guan, Zehuan Wang, Xiaoying Jia, Xipeng Li, Minyi Guo, and Yuhao Zhu. Accelerating sparse dnn models with-

out hardware-support via tile-wise sparsity. *arXiv preprint arXiv:2008.13006*, 2020. 1

[7] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 2, 3

[8] Maxwell Horton, Yanzi Jin, Ali Farhadi, and Mohammad Rastegari. Layer-wise data-free cnn compression. *arXiv preprint arXiv:2011.09058*, 2020. 2, 5, 7

[9] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020. 2, 5, 6

[10] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018. 6

[11] Hirokatsu Kataoka, Kazushige Okayasu, Asato Matsumoto, Eisuke Yamagata, Ryosuke Yamada, Nakamasa Inoue, Akio Nakamura, and Yutaka Satoh. Pre-training without natural images. In *Proceedings of the Asian Conference on Computer Vision*, 2020. 7

[12] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pages 5544–5555. PMLR, 2020. 3

[13] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. A deeper look at the layerwise sparsity of magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020. 2, 3

[14] Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European Conference on Computer Vision*, pages 639–654. Springer, 2020. 5

[15] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021. 2, 5

[16] Xiao Liu, Wenbin Li, Jing Huo, Lili Yao, and Yang Gao. Layerwise sparse coding for pruned deep neural networks with extreme compression ratio. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4900–4907, 2020. 1

[17] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020. 2, 5

[18] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019. 2, 4

[19] Kodai Nakashima, Hirokatsu Kataoka, Asato Matsumoto, Kenji Iwata, and Nakamasa Inoue. Can vision transformers learn without natural images? *arXiv preprint arXiv:2103.13023*, 2021. 7

[20] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33, 2020. 2

[21] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015. 2

[22] Michael R Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. *arXiv preprint arXiv:1907.08610*, 2019. 6

[23] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 2