An Empirical Study of Pruning and Quantization Methods for Neural Networks

PETER FORSYTH and RAPHAEL TANG and ZEHAO XU
University of Waterloo

We experiment with compressing neural models via pruning and quantization. We find that when pruning is performed without retraining, the HCS-LASSO method dominates, with the caveat that it requires a subset of the training data. We find that most methods, including random pruning, perform similarly when one-shot retraining is allowed, calling into question the justification of pruning methods by their performance with retraining. In contrast, Taylor pruning beats random pruning on a slowly-pruned transferred model. Finally, we experiment with quantization, replicating a number of results and confirming the effectiveness of several techniques.

1. INTRODUCTION

The popularity of smart routers, smartphones, and smartwatches, creates a growing need to perform intelligent tasks on low-power computers. Deep neural networks attain state-of-the-art performance on many of these tasks, such as speech recognition, face detection, and text prediction; however, the computational requirements of large networks are prohibitive. For example, AlexNet [Krizhevsky et al. 2012] uses 725M floating-point operations (FLOPs) in a single forward pass. More recent architectures like ResNet [He et al. 2016] and GoogLeNet [Szegedy et al. 2015] are even more heavyweight. Thus, compressing large models is an important problem for the deployment of neural models.

One method for neural network compression is to zero out unnecessary weights, according to some criterion. Naturally, null weights can then be discarded, resulting in a network with a smaller memory footprint. Methods that follow this paradigm are said to be "pruning" the neural network. In the present work, we empirically evaluate pruning approaches by Louizos et al. [2017], Han et al. [2015a], He et al. [2017], Liu et al. [2017], and Li et al. [2016a].

Another method for compression is to use fewer bits for representing weights or activations; this paradigm is called quantization. In the extreme case, single-bit quantization results in values of ± 1 [Hubara et al. 2016b]. We examined quantization as described in the texts by Hubara et al. [2016a] and Li et al. [2016b].

We developed a Pytorch [Paszke et al. 2017] framework for model compression, which we use for all our experiments.

2. PRUNING

Because deep neural networks are redundant, many of their weights can be set to zero with minimal effect on performance [Han et al. 2015b]. While procedures that set weights to zero in an unstructured way reduce a network's storage and memory consumption, they do not reduce its computational cost, since neural networks are implemented using dense linear algebra libraries that cannot exploit unstructured sparsity. In contrast, we study methods for zeroing entire input or output channels of convolutional layers. Such methods remove rows or columns from the underlying matrices [Chetlur et al. 2014], reducing computational cost.

2.1 Pruning Techniques

2.1.1 *Channel Norm Pruning.* This strategy, explored by Li et al. [2016a], iteratively prunes the channel whose weights have the smallest norm in a given layer. The idea is that channels with small weight-norm are unlikely to be decisive in the final classification. We consider two approaches for applying channel norm pruning to the entire network. In the first, a fixed proportion of channels are pruned from each layer. In the second, following Molchanov et al. [2016], we score each channel by diving its norm by the sum of the norms of all channels in the same layer. Then we prune the channels with the globally smallest scores across the network.

2.1.2 *Random Pruning.* Explored by Molachanov et al. [2016] and Mittal et al. [2018], this approach zeros random channels of the layers to which it is applied. We prune a fixed proportion of the channels in each layer.

2.1.3 *HZS-LASSO.* To reduce the number of input channels in a given convolutional layer from $c_i \in \mathbb{Z}_+$ to $c_i' \in \mathbb{Z}_+$, this procedure, proposed by He et al. [2017], uses two stages. Let $Y \in \mathbb{R}^{c_o w_o^2 b}$ be the vector whose entries store the layer's output at each of $c_o \in \mathbb{Z}_+$ output channels and $w_0^2 \in \mathbb{Z}_+$ spatial locations for every image in a training batch of size $b \in \mathbb{Z}_+$. Let $A \in \mathbb{R}^{b w_o^2 c_o \times c_i}$ be the matrix whose entries are the same outputs separated according to the contribution of each input channel. Thus summing the rows of A yields Y. In the first stage of the procedure, we find a function $\beta : \mathbb{R}_+ \to \mathbb{R}^{c_i}$ satisfying

$$\beta(\lambda) \in \underset{\beta \in \mathbb{R}^{c_i}}{\operatorname{argmin}} \|Y - A\beta\|^2 + \lambda \|\beta\|_1 \tag{1}$$

by tracing the LASSO regularization path (See Friedman et al. [2001] chapters 3.4 and 3.8. We use Scikit-learn's `lasso_path` function [Pedregosa et al. 2011].). We then select $\lambda^* := \inf\{\lambda \in \mathbb{R}_+ : \|\beta(\lambda)\|_0 \leq c_i'\}$ and $\beta^* := \beta(\lambda^*)$. We prune input channels corresponding to zero entries of β^*. In the second stage we find new weights for the pruned convolutional layer by minimizing the Euclidean norm of the difference between the output of the full convolutional layer and the output of the pruned convolutional layer over a subset of the training set. In this way, we seek to prune input channels with minimal change to the output of the convolutional layer. We repeat this process for each layer, pruning the same proportion of weights in each. HZS-LASSO requires training data, unlike the other methods we consider, but this data need not be the entire training set: in our experiments we use 500 images from a training set of 50000.

2.1.4 *Network Slimming.* This approach, described in Liu et al. [2017], adds a penalty term to the loss function during training equal to a weight times the sum of the absolute values of all the batch-normalization scale factors in the network. At the end of training, those channels corresponding to the smallest batch-norm scale factors are pruned. Intuitively, since batch normalization results in each channel having approximately zero mean and unit variance across a batch, channels multiplied by small scale factors after batch normalization are likely insignificant. To verify our implementation, we closely reproduced some results from Liu et al. [2017] on Resnet-164.

2.1.5 *Taylor.* Let $(h_i)_{i \in I}$ denote the scalar outputs produced by a given channel of a convolutional layer across all spatial locations. The magnitude of the effect on the loss function $\mathcal{L}$ of setting this channel to zero can be approximated by first-order differential approximation as $|\sum_{i \in I} h_i \frac{\partial \mathcal{L}}{\partial h_i}|$. The Taylor pruning approach proposed by Molchanov et al. [2016] assigns a score to every output channel of a convolutional layer equal to its $|\sum_{i \in I} h_i \frac{\partial \mathcal{L}}{\partial h_i}|$. The score is computed image-by-image, and then averaged over a subset of the images in the training set. Those channels with the lowest score are pruned. The idea is to prune channels whose components have a relatively small effect on the final classification. The derivatives $\frac{\partial \mathcal{L}}{\partial h}$ are computed during normal back-propagation, and consequently this approach is computationally cheap. To facilitate the comparison of scores between layers, the authors divide the score of every channel by the square root of the sum of the squares of the scores of all channels in the same layer. We use the same approach.

2.1.6 *Stochastic l_0 Regularization.* During training Louizos et al. [2017] multiply each output channel of a convolutional layer by a random variable, which serves as a gate. Each gate is distributed according to the *Hard Concrete* distribution [Maddison et al. 2016], which approximates the Bernoulli distribution, but has nonzero density in $(0, 1)$. A parameter α, which controls the frequency with which the gate is on, is learned during training, and a term is added to the loss function penalizing gates that are frequently on. At test time, the probabilistic gates are converted to deterministic multipliers based on the values of α. Channels with zero multipliers are pruned. To verify our implementation of this complicated technique, we replicated the settings of some experiments on MNIST performed in the paper. Our results closely resemble those described by Louizos et al. [2017], and we attribute the differences to the highly stochastic nature of this scheme.

3. QUANTIZATION

Traditionally, weights and activations are represented using 32-bit floating point numbers. However, as past work has shown, such precision is not required for good performance in neural inference; most of the weights tend to be "close" to zero, suggesting they can be quantized to fewer bits. In the extreme case, weights and activations have been quantized to ± 1, resulting in massive memory and computational savings.

In this work, we explore BinaryConnect [Courbariaux et al. 2015], binarized neural networks (BNNs) [Hubara et al. 2016a], and ternary weight networks (TWNs) [Li et al. 2016b].

3.1 Quantization Techniques

3.1.1 *BinaryConnect.* In BinaryConnect [Courbariaux et al. 2015], weights are limited to ± 1. Specifically, the floating point weights w are passed through a quantization function $\hat{w} = f(w) = 2\mathbb{I}_{\geq 0}(w) - 1$ before being used in the layers, where $\mathbb{I}$ is the indicator function. In order to backpropagate through this non-differentiable function, the straight-through estimator [Hinton 2012] is used:

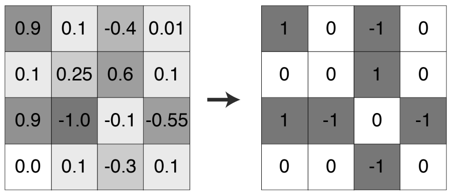

Fig. 1. Ternary quantization

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{w}}$$

Additionally, floating point weights are clipped to $[-1, 1]$ after each optimization step. Since the weights are extremely large by neural network standards, the authors apply batch normalization after every binary layer.

3.1.2 *Binarized neural networks.* In BNNs, Hubara et al. [2016a] propose binarizing activations in addition to weights. They use "binary tanh," a binary activation function:

$$\text{BinTanh}(x) = 2\mathbb{I}_{\geq 0}(x) - 1$$

$$\frac{\partial \text{BinTanh}}{\partial x} = \begin{cases} 1 & \text{if} -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

The paper also derives a binary version of batch normalization, where nearly all multiplications are approximated by bit-shifts instead (power of 2 multiplication). Surprisingly, binary batch norm results in no perceived loss in accuracy.

3.1.3 *Ternary weight networks.* TWNs allow for weights to be one of $\{-1, 0, 1\}$. Instead of approaching this problem heuristically as in BinaryConnect, Li et al. (2016) explicitly solve $\text{argmin}_{\alpha, \Delta} \|W - \alpha \hat{W}\|_2^2$, where

$$\hat{W}_{ij} = \begin{cases} -1 & \text{if } \hat{W}_{ij} \leq -\Delta \\ 1 & \text{if } \hat{W}_{ij} \geq \Delta \\ 0 & \text{otherwise} \end{cases}$$

Empirically, they find that W is roughly normally or uniformly distributed, so they choose α and Δ accordingly.

3.1.4 *8-bit linear quantization.* A real matrix $W \in \mathbb{R}^{m \times n}$ is said to be linearly quantized if there exists an affine mapping

$$W = a(\hat{W} + b)$$

where $\hat{W} \in \mathbb{Z}_u^{m \times n}$, $\mathbb{Z}_u = \{0, 1, \ldots, 255\}$. In practice, libraries like `gemmlowp`[1] offer efficient support for 8-bit quantized matrix multiplication. We applied this scheme to the weights.

[1]https://github.com/google/gemmlowp

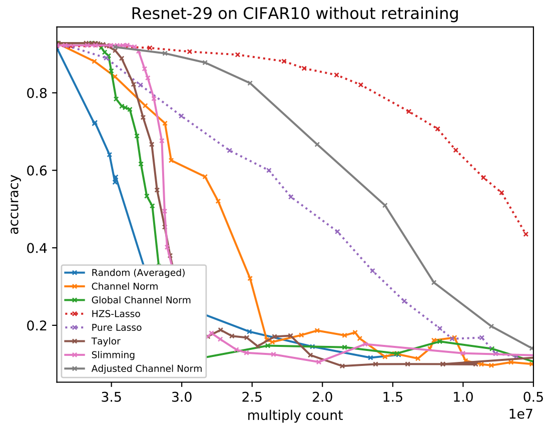

Fig. 2. Accuracy as a function of forward-pass multiplies. The reported accuracy for random pruning is averaged over 100 random choices of channels to prune.

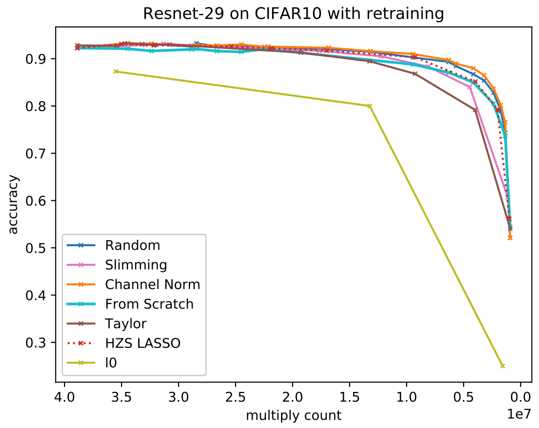

Fig. 3. Accuracy as a function of forward-pass multiplies.

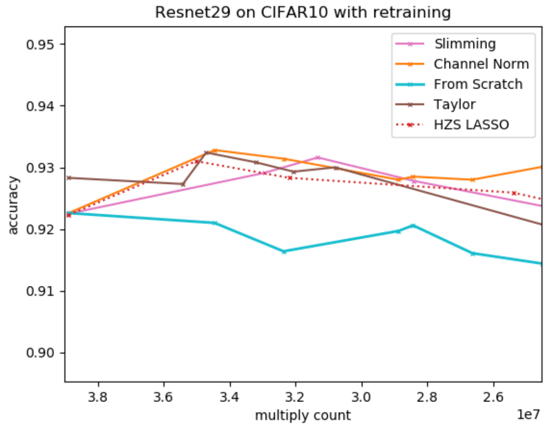

Fig. 4. Accuracy as a function of forward-pass multiplies (zoomed in).

4. EXPERIMENTS

4.1 Pruning Experiments and Discussion

4.1.1 *Without Retraining.* We begin with Resnet-29 [He et al. 2016] trained for 100 epochs on CIFAR10 [Krizhevsky and Hinton 2009] with the cosine annealing learning rate schedule described by Loshchilov and Hutter [2016] and an initial learning rate of 0.2. We use weight decay of 0.0001 and momentum of 0.9. In addition to standard cropping and horizontal flipping data augmentation, we also use random erasure as described by Zhong et al. [2017]. We simply apply the pruning techniques to the trained model and report the resulting accuracy as a function of the number of floating-point multiplies performed in a forward pass.

As Figure 2 shows, the global methods (Taylor, slimming, and global channel norm) achieve high accuracy at low levels of pruning, but dramatically worsen around $3.25 \times (10)^7$ multiplies. One possible explanation is that the global methods greedily select channels which separately are of little importance but collectively matter a great deal, causing a collapse in accuracy when a significant proportion of these channels are pruned. In the case of Taylor pruning, we can also argue that as more channels need to be pruned, the difference between the activations of pruned network and the original network reach a point at which the Taylor approximation becomes useless. To further investigate the precipitous decline in accuracy of the global methods, we recorded which channels were pruned around the dramatic accuracy drop. We discovered that at the point of the drop, many channels from the final layers of residual blocks were pruned. Hoping to use this to improve the Taylor method, we implemented a version that simply never pruned from final residual layers. However, this version performed even worse, suggesting that careful hand-tuning of layer priorities would be required to develop a pruning method in this vein.

Figure 2 also shows the superiority of the HZS-LASSO method. It is natural to ask whether either the first stage, in which the channels to prune are selected by LASSO regularization, or the second stage, in which least-squares is used to adjust the weights of the pruned convolutions, is key to the performance of the method. To explore this question, we experimented with applying least-squares weight adjustment to a network pruned by channel-norm pruning. As the line labeled “Adjusted Channel Norm” shows, the resulting method is better than basic channel-norm pruning but worse than HZS-LASSO. We also experimented with using only the first stage of the HZS-LASSO procedure. That is, we solved for β^*, pruned channels corresponding to zero entries of β^*, and re-scaled the weights of the other channels by their corresponding entry in β^*. The method, labeled ”Pure LASSO”, performs well, but worse than HZS-LASSO. Together these results indicate that both components of the HZS-LASSO method are important: LASSO regularization selects channels that can be pruned, and least squares weight adjustment partially compensates for their pruning.

4.1.2 *With Retraining.* We use Resnet-29 trained with the same settings as above. We then restart training, and apply the pruning technique over the course of 10 epochs (except for HZS-LASSO, which we apply in one-shot). We continue to train the model for 90 epochs, and report the resulting accuracy. Figures 3

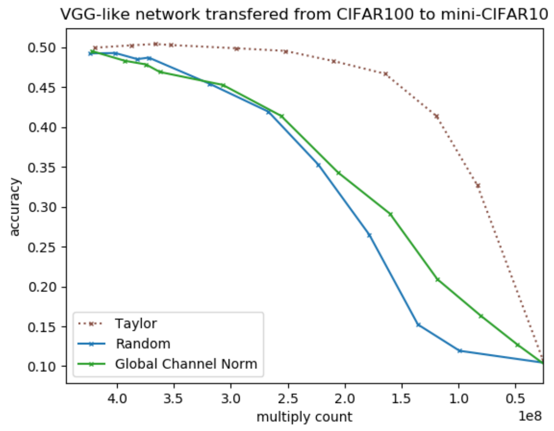

Fig. 5. Accuracy as a function of forward-pass multiplies. Accuracies for Taylor and Channel Norm are averaged over 5 runs, while accuracies for Random are averaged over 10 runs.

and 4 show the results. For comparison, we also plot the accuracy of a neural network with the same architecture as the channel-norm pruned network, but trained from scratch with randomized initial weights. A first observation is that for low levels of pruning, all methods beat the from-scratch baseline, suggesting that after pruning, information is retained from the larger model. A second observation is that, unlike in the no-retraining case, here the accuracies of almost all methods are close, and remain high at low and moderate degrees of pruning. Indeed, random pruning is the second-best performing method at high levels of pruning. This suggests that some recent papers, which devise pruning methods and then offer the performance of the pruning method after retraining as evidence of its efficacy (e.g. [Liu et al. 2017]), are in error: most methods perform similarly after retraining. The finding that random pruning performs well after retraining agrees with the recent paper of Mittal et al. [2018], who argue that this a consequence of the inherent redundancy of neural networks, and is analogous to the recovery of a patient after brain injury. Lastly, we discuss the l_0 method of Louizos et al. [2017]. We decided to compare this unique method to the with-retraining methods, since to adjust its degree of pruning, one must retrain the entire network. In this comparison, the method performed poorly in out experiments, and was not competitive with much simpler methods. We experimented with applying different degrees of regularization to different layers within a residual block, or across residual blocks, and with only applying stochastic gates to certain layers. None of these approaches yielded appreciable improvement in the l_0 method.

4.1.3 *Transfer Learning.* The results of the previous section establish that, for the Resnet-29 model we investigated, when retraining is feasible, the choice of pruning method is not extremely significant. While few papers conduct detailed comparisons of channel pruning methods, Molchanov et al. [2016] do so in the context of transfer learning, wherein a model trained to perform one task on a large dataset is adapted to perform a different but related task on a smaller dataset, and then pruned. They find that the choice of pruning method is quite significant, in contrast to our results in the previous section. To provide evidence supporting or contradicting their findings, we conducted a small empirical study of transfer learning. Molchanov et al. [2016] transfer from ImageNet to a dataset of bird photos. Since we lack the time to train om ImageNet, we instead transfer from CIFAR100 to a dataset we call *MINICIFAR10*. MINICIFAR10 has the same test set as CIFAR10, and has a validation set consisting of 1000 CIFAR10 images, but has only 200 images in its training set[2]. We train a VGG-like network on CIFAR100 for 100 epochs via SGD with a learning rate of 0.03, momentum of 0.9, and weight decay of 0.0001. We then replace the final layer of the network with one with 10 outputs instead of 100, train the network for 200 more epochs on MINICIFAR10, then alternate between pruning one channel and training one epoch until the desired degree of pruning is reached, then train for a further 200 epochs. We report the results in Figure 5. Unlike in the previous experiment, here a sophisticated method (Taylor), is able to soundly beat random pruning.

Method	CIFAR-10 Accuracy	CIFAR-100 Accuracy
BinaryConnect	90.05%	68.41%
BNNs	87.88%	66.91%
TWNs	92.7%	70.36%
8-bit	92.92%	70.85%
Vanilla	93.21%	71.16%

Table I.
Quantization

4.2 Quantization Experiments

We report results on CIFAR-10 and CIFAR-100. We use a variant of VGG-16 described in Hubara et al.'s work [Hubara et al. 2016a]. We used a learning rate of 0.1, which is reduced by a factor of 10 if validation accuracy does not improve for 15 epochs. The results align well with the intuition behind the techniques.

5. CONCLUSION

When some training data is available but resource constraints prevent retraining via backpropagation, we recommend the HZS-LASSO pruning method. When retraining is possible and the task for the pruned model is the same as for the original model, the choice of pruning method seems not to matter much. To perform transfer learning with slow pruning, the Taylor method is a good choice. Quantization schemes can improve the efficiency of deep neural networks without much loss of accuracy, although they make training slower and more difficult.

A question raised by our work is that of the relationship between the results of sections 4.1.2 and 4.1.3. Can the contrast between the results be explained entirely by the slow pruning schedule used in 4.1.3, which probably improves the Taylor method by making the first-order approximation more accurate? Or is transfer learning a favorable setting for pruning, perhaps because sophisticated methods can identify channels relevant to the old task but not the new task. We performed preliminary experiments that may indicate that both factors are operative, but further investigation is required.

[2]A VGG-like model trained for 1000 epochs on this dataset without transfer learning dos not learn anything, achieving an accuracy of about 0.1

REFERENCES

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. 1. Springer series in statistics New York.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

Song Han, Jeff Pool, John Tran, and William Dally. 2015b. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. *Second International Conference on Computer Vision* (2017), 1389–1397.

Geoffrey Hinton. 2012. Neural Networks for Machine Learning. Lecture 9. (2012).

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016a. Binarized neural networks. In *Advances in neural information processing systems*. 4107–4115.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016b. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061* (2016).

Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

Fengfu Li, Bo Zhang, and Bin Liu. 2016b. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016a. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2755–2763.

Ilya Loshchilov and Frank Hutter. 2016. SGDR: stochastic gradient descent with restarts. *Learning* 10 (2016), 3.

Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning Sparse Neural Networks through L_0 Regularization. *arXiv preprint arXiv:1712.01312* (2017).

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* (2016).

Deepak Mittal, Shweta Bhardwaj, Mitesh M Khapra, and Balaraman Ravindran. 2018. Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks. *arXiv preprint arXiv:1801.10447* (2018).

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. (2016).

Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration. (2017).

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, and others. 2015. Going deeper with convolutions. Cvpr.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2017. Random Erasing Data Augmentation. *arXiv preprint arXiv:1708.04896* (2017).