Targeted and Automatic Deep Neural Networks Optimization for Edge Computing

Luca Giovannesi[0009-0002-4214-0493], Gabriele Proietti Mattia[0000-0003-4551-7567], and Roberto Beraldi[0000-0002-9731-6321]

[1] Department of Computer, Control and Management Engineering "Antonio Ruberti", Sapienza University of Rome
`https://www.diag.uniroma1.it`
[2] `{giovannesi,proiettimattia,beraldi}@diag.uniroma1.it`

Abstract. DNNs, commonly employed for complex tasks such as image and language processing, are increasingly sought for deployment on Internet of Things (IoT) devices. These devices operate with constrained resources, including limited computational power, memory, slower processors, and restricted energy requirements. Consequently, optimizing DNN models becomes crucial to minimize memory usage and computational time. However, traditional optimization methods require skilled professionals to manually fine-tune hyperparameters, striking a balance between efficiency and accuracy. This paper introduces an innovative solution for identifying optimal hyperparameters, focusing on the application of pruning, clusterization, and quantization.
Initial empirical analyses were conducted to understand the relationships between model size, accuracy, pruning rate, and the number of clusters. Building upon these findings, we developed a framework that proposes two algorithms: one for discovering optimal pruning and the second for determining the optimal number of clusters. Through the adoption of efficient algorithms and the best quantization configuration, our tool integrates an optimization procedure that successfully reduces model size and inference time. The optimized models generated exhibit results comparable to, and in some cases surpass, those of more complex state-of-the-art approaches.
The framework successfully optimized ResNet50, reducing the model size by 6.35x with a speedup of 2.91x, while only sacrificing 0.87% of the original accuracy.

Keywords: Deep Neural Networks; DNN Acceleration; DNN Compression; Edge Computing

1 Introduction

Deep Neural Networks (DNNs) have garnered increasing popularity over the past decade due to their adeptness in addressing intricate problems such as image recognition, language processing, signal processing, and more. Simultaneously, the rapid proliferation of the Internet of Things (IoT) has led to an increasing demand for deploying DNN models on IoT devices.

However, while these devices offer the advantage of easy installation in numerous contexts where traditional servers may be impractical, this convenience comes at a price. IoT devices typically feature constrained computational power, limited memory capacity, and comparatively slower processors in comparison to conventional desktops and laptops. Furthermore, a substantial number of these devices operate on battery power, introducing additional constraints related to energy consumption. Usually, DNNs require a powerful GPU to run effectively, a component that is generally not available in most IoT devices, and therefore, here emerges the need to optimize neural network models to reduce memory usage and the computational time required for the inference operation.

Nowadays, the most common approaches for neural network optimization require heuristic hand-tuning of some fundamental hyper-parameters by an expert in the field, whose goal is to find the optimal balance point between model optimization and model accuracy degradation. In this paper, we present a tool that is able to automatically find these hyper-parameters and use them to apply the optimizations, which are pruning, weight clustering and quantization.

The rest of the paper is organized as follows. In Section 2 we give some background to the reader about the techniques that are used for compressing DNN models, in Section 3 we present different related work to our proposed solution, in Section 4 we describe the algorithms used in the tool and, finally, in Section 6 we draw the conclusions.

2 DNN compression techniques

2.1 Network pruning

DNNs exhibit notable redundancy in their parameterization, containing portions that are not truly essential [1]. Recognizing this phenomenon, we can eliminate some of these redundant parts to create smaller and simpler models. However, when removing a portion of the DNN, the accuracy typically decreases. The key challenge lies in identifying the most suitable parts of the network to be pruned while minimizing the decrease in accuracy.

Network pruning can be categorized into three main types: Channel pruning, Filter pruning, and Connection pruning.

Channel pruning The idea behind channel pruning is to decrease the number of input and/or output channels in convolutional layers without compromising performance. In this context, the authors of [2] achieved a 5x speedup on VGG-16 [3]. However, on modern networks such as ResNet [4] and Xception [3], the acceleration is only 1.4%.

Filter pruning The Convolutional Neural Network (CNN) employs a substantial number of filters to enhance precision. However, each filter adds the overall number of floating-point operations, thereby increasing the latency of the network. The goal of filter pruning is to eliminate less crucial filters. In [5], the authors achieved a reduction in floating-point operations by approximately 3.2 and 1.58 times for VGG-16 and ResNet-50, respectively, while maintaining nearly the same level of accuracy.

Connection pruning The size of a DNN is dictated by the number of connections between its layers. A larger number of parameters result in a bigger model, requiring more operations. The concept behind this pruning technique is to eliminate unimportant connections.

A straightforward yet effective approach is Global Magnitude Pruning (MP) [6], where all weights with absolute values below a predefined threshold are removed.

2.2 Weight sharing

In weight sharing, also called weight clustering [7], only a small subset of values are used to represent all the weights of the model.

For the creation of the clusters different algorithms can be used, in [8] the weights distribution is based on a low-cost hash function, in which all connections with the same hash share a single parameter value.

Another approach is the one proposed by Han et el. [9], where k-means algorithm is used for the clusterization, in each iteration it assigns the values of the weights to the clusters with the nearest centroid, it converges when no addition assignments are needed. The time required for the convergence depends by the initial value of the centroids, the most used method is using the k-means++ algorithm [10].

2.3 Quantization

Quantization [11] reduces the bit width of weights and activation functions, typically from 32 bits (FP32) to 8 bits (INT8/UINT8). This compression yields advantages such as a 4x reduction in memory overhead and a quadratic decrease in computational cost for matrix multiplication by a factor of 16.

Quantization can be applied to weights only or both activation functions and weights. While NNs are generally robust to quantization, when a low bit-width quantization (< 8 bits) is employed, noise is introduced, potentially decreasing accuracy. Robustness varies across networks, necessitating additional efforts to leverage quantization benefits.

The two methods for quantizing neural networks are Post Training Quantization (PTQ) and Quantization Aware Training (QAT).

Post Training Quantization (PTQ) In PTQ a pre-trained FP32 neural network is converted directly to an integer model without additional tuning. If some sample data are provided, they may help the calibration of the parameters. PTQ can be applied on both weights and activation functions using 8 bits integers. It can be applied only weights keeping the original floating-point activation functions. Otherwise a combination such as int8 weights and int16 activation functions. Alternatively, the entire model can be quantized to 16-bit floating point. Our experiments (Table 1) demonstrate the impact of various quantization approaches on model size and inference time.

Quantized elements	**Accuracy**	**Model size (MB)**	**Inference time (ms)**
Weights fp32 (Original) Activation functions fp32	0.832	12.28	9.240
Weights int8 Activation functions int8	0.821	3.18	4.055
Weights int8 Activation functions fp32	0.820	3.18	4.200
Weights int8 Activation functions int16	0.820	3.32	277.52
Weights fp16 Activation functions fp16	0.832	6.26	9.240

Table 1: MobileNet V2 quantized with different configurations

Using int8 for weights and int16 for activation functions (4th row in Tab. 1) results in slower inference due to incomplete support by the kernel, as noted in TensorFlow documentation. The FP16 model maintains the original inference time while reducing the model size by half.

Quantization Aware Training (QAT) PTQ is fast to implement, it does not require labeled data, and in most of the cases the quantized model has an accuracy close to the original model, however, when the bit width is very low, like 4 bits, the PTQ introduces too many errors.

In QAT some nodes are added to the network in order to simulate the error introduced by the quantization, and then the model is trained with the presence of the added nodes improving the robustness to the quantization noise. In general, QAT provides better performance but requires a labeled dataset and fine-tuning which takes computational time.

Figure 1 illustrates the weight distribution of the mentioned optimization approaches.

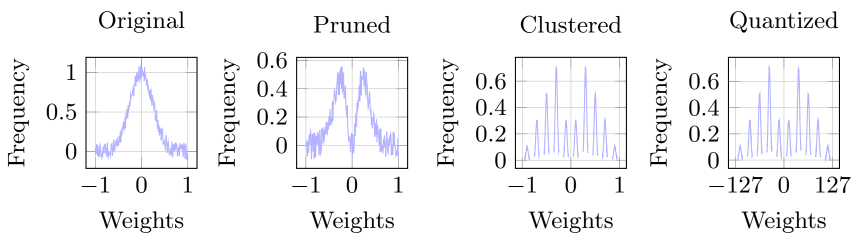

Fig. 1: Weight distribution in the different steps of the described optimization methods

3 Related work

In the literature, a lot of works use the illustrated techniques for model compression, but only a few proposals find automatically the hyper-parameters necessary to apply them. Regarding the pruning, the necessary hyper-parameter is the pruning rate, for the clustering is the number of clusters, while for the quantization the searching space is about the optimal configuration between the ones presented in Table 1.

3.1 Pruning rate search

In [12] the authors claim that if the absolute values of the weights of each layer are placed in order, they are very similar to each other, approximating a function in which in the first part the values of the weights increase linearly, while in the second part, the values increase exponentially, these two parts are divided by a point called "demarcation point". They find a correlation between the threshold used for Global Magnitude Pruning manually chosen in [9] and the demarcation point, claiming that usually the optimal threshold is near this point. However they have reported this correlation only for Lenet-5 [13], AlexNet [14], VGG [15], additional tests are necessary to check if the correlation is always present.

In Optimal Brain Surgeon (OBS) [16], a method determines which weights to remove based on their impact on loss function L. This often involves inverting the Hessian matrix $\mathbf{H} = \partial^2 L/\partial^2 W$, a very hard task with modern neural networks that have millions of parameters W.

L-OBS [17] addresses this by applying OBS to individual layers and proposing an algorithm to reduce the Hessian matrix for feasible inversion. NAP [18] uses an efficient approximation of the Hessian, utilizing a Kronecker-factored Approximate Curvature method. However, these methods provide a lower compression rate than methods based on weights magnitude pruning.

The authors of [19] propose a reinforcement learning approach, in which the agent receives layer characteristics and returns the pruning rates for each layer, which will be used to prune the model. Results are similar or better to manual methods on VGG-16[15] and ResNet 50 [4].

Runtime Neural Pruning [20] (RL based approach), dynamically optimizes models during run-time adjusting the balance point based on input data, the original model is always preserved for restoration.

The solutions in which the hyper-parameter search is solved as optimization problem [21,22] provide a higher precision, but this kind of method has large computational costs slowing the convergence time.

3.2 Number of clusters search

In [23] LetNet-5 [13] is clustered using K-Means with a greedy algorithm for cluster number selection, However, this approach is slow for larger models. Instead, in [24] authors solve the problem as a minimization task, but the computational cost tends to be high despite optimal solutions.

To the best of our knowledge, the only relevant work in which all the previous optimizations are used together is the one by Song Han et al. [9], but there is not evidence of methods to automatically assign the hyperparameters.

4 Proposed method

The proposed framework receives a trained deep neural network model, a dataset, and additional parameters as input. It then autonomously identifies the optimal pruning rate, followed by determining the optimal number of clusters, and ultimately, finding the best configuration for quantization.

4.1 Weight pruning

Our idea uses Global Magnitude Pruning (MP), despite the simplicity of this method, it is possible to get results comparable to the state-of-the-art [25,26], with respect also to complicated pruning algorithms present in the literature.

Global Magnitude Pruning (MP) is a method where weights with an absolute magnitude value below a specified threshold are eliminated. The threshold is set to achieve a predetermined sparsity rate after the pruning process. Through experiments, an inversely proportional relationship between the pruning rate and accuracy was observed, as depicted in Figure 2. This finding enables the use of binary search to determine the pruning rate that yields the desired accuracy.

In the searching algorithm, it is allowed an error of ϵ between the desired accuracy and the reached accuracy. In our case, the desired accuracy is slightly less than the one initially measured on the original model. Looking for a smaller accuracy guarantees that the returned value is the one before the quick accuracy decreases.

Binary search converges since both the pruning rate and the measured accuracy are linear, this implies that changing the pruning rate is always possible to reach the desired accuracy.

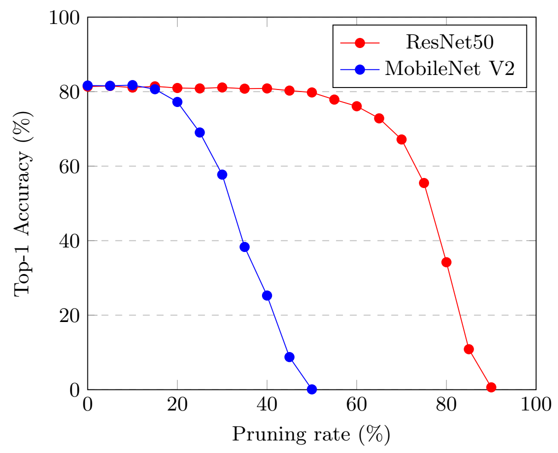

Fig. 2: Accuracy variation with different pruning rates for ResNet50 and MobileNet V2

4.2 Number of clusters

In order to minimize the model size, we have to minimize the number of clusters, however a small number of clusters cannot be sufficient to reach the original model accuracy.

In our experiments, MobileNetV2 was clustered with different numbers of clusters. The results are visible in figure 3 in which there is an evident a direct correlation between the model size and the number of clusters, while there isn't any evident relation between the number of clusters and the inference time. Therefore the optimal amount of clusters is the smaller one in which is possible to reach the desired accuracy, in other words, given an original model $M_{original}$, the pruned model M_{pr}, a function $CL(M, n_{cl})$ which clusters a model M with n_{cl} clusters and a ϵ defined as the maximum allowed difference between the original model accuracy and the clusterized model accuracy. The optimal number of clusters cl_{opt} are:

$$cl_{opt} = \underset{cl}{\operatorname{argmin}} \, |Acc(M_{original}) - Acc(CL(M_{pr}, cl)| < \epsilon \tag{1}$$

Then giving the function f defined as:

$$f(x) = \begin{cases} 1, & \text{if } |Acc(M_{original}) - Acc(CL(M_{pr}, cl)| < \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Which returns 1 if the given number of clusters x returns a model that matches the required accuracy constraints, 0 otherwise, from the analysis we know that $f(x) \leq f(x+1)$, then to find the optimal number of clusters we have to find a x such that $f(x) < f(x+1)$. The problem of finding the optimal value of x can be solved using an algorithm based on ternary search, so given a range of clusters we have to find the step (the transaction between 0 and 1). Then for each iteration the range is split in 3 intervals by two points, and computing the values of these points is possible to determine in which segment the step belongs, then the algorithm is applied again in the new segment, it ends when the difference between the two points is 1.

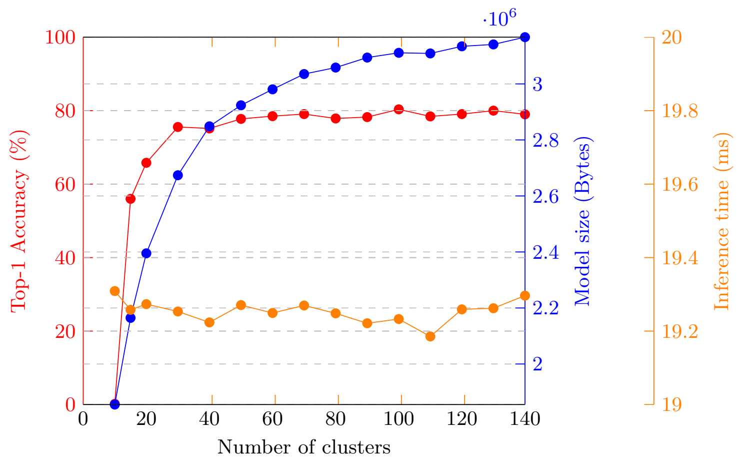

Fig. 3: Metrics affected with different number of clusters on MobileNet V2

4.3 Quantization

The used technique for quantization is the PTQ, because, differently from QAT, it does not require fine-tuning and the accuracy loss is negligible until we don't go below 8 bits. Quantization affects models differently; what works for one may not for another.

The efficiency of PTQ allows testing various methods outlined in Table 1 without significant time overhead, methods are tested from the more aggressive quantization for faster but less accurate models to softer quantization for slower but more accurate models, using this rank, the first method that matches the original desired accuracy is selected.

From the proposed methods (ones used in Tab. 1) the one that uses int8 for weights and int16 for activation functions is excluded, since as reported in the documentation of TensorFlow Lite (the DNN framework used for our tests) the lack of implementation brings slow inference time.

4.4 Overall procedure

This section describes the full procedure, including all of the previous techniques. Figure 4 illustrates a flowchart depiction. Initially, the program examines the input model and establishes an accuracy baseline for the next operation. Then it

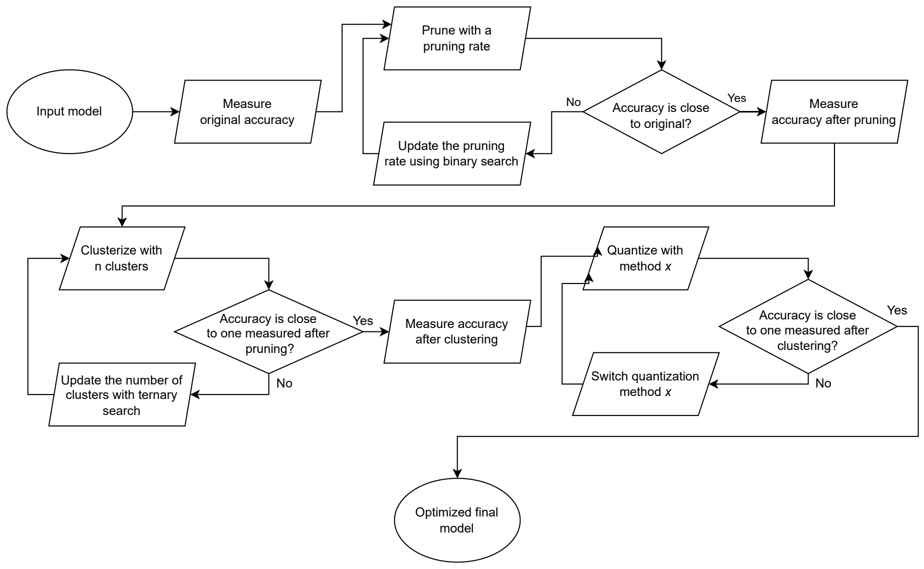
Input model
Measure original accuracy
Prune with a pruning rate
Accuracy is close to original?
No
Yes
Update the pruning rate using binary search
Measure accuracy after pruning
Clusterize with n clusters
Accuracy is close to one measured after pruning?
Yes
No
Update the number of clusters with ternary search
Measure accuracy after clustering
Quantize with method *x*
Accuracy is close to one measured after clustering?
Yes
No
Switch quantization method *x*
Optimized final model

Fig. 4: Flowchart representing the overall optimization procedure

starts to prune the network with its methods. After experimenting with several hyper-parameters, it returns the best pruning configuration. Next, the pruned model is evaluated to establish a new baseline, thus weight clustering is used, and the program runs the algorithm designed specifically to calculate the ideal number of clusters of the pruned model, returning the optimal one. Finally, after establishing a new baseline from the clustered model, it tries multiple quantization algorithms, ranking them from more aggressive to kinder, and returns the first that meets the required target accuracy. At this stage, the software builds the optimized model based on all of the previously determined optimal parameters.

5 Results

Making a comparison with the literature is not easy, since the proposed software can optimize different models used to solve different problems. Furthermore, the literature employs diverse metrics: some use inference times for speedup calculation, and others focus on FLOPs or MAC operations reduction. Regarding model compression, metrics range from parameter count reduction to actual model size reduction.

Our evaluation metrics are the speedup factor, calculated as the ratio of the inference times between original and optimized models, and the compression rate, calculated as the ratio of the original model's size to the optimized one.

We remark by adjusting ϵ values in pruning and clustering algorithms, models with varying accuracies can be achieved.

Model	**Compression rate**	**Speed up**	**Δ Top1**
Our	12.95x	4.77x	0.9%
Our (lower ϵ)	12.19x	3.40x	0.26%
Play and Prune [21]	13.0x	-	0.3%

Table 2: VGG16 on CIFAR10 comparison, the model is optimized with our tool also with different ϵ values for the pruning and clusterization algorithms.

Method	**Speedup**	**Compression rate**	**Δ Top1**
Our	**2.91x**	**6.35x**	**0.87%**
NAP[18]	2.3x	-	2%
Thinet [5]	2.3x	-	4.3%
CPI [2]	2x	-	3.3%
AMC [19]	-	5x	0.02%
LWC [6]	-	2.7x	0.01%

Table 3: ResNet50 model optimization comparison.

The listed results are executed on different hardware and as reported in other works, the speedup changes with different hardware, it decreases when the device parallelism is higher. Since most of the competitor projects have the source code not available, we are not able to run the proposed frameworks in order to run the optimized models on the same hardware, so the data reported is the one written in the related articles.

Regarding the compression rate, we remark that it does not depend on the hardware and it can be comparable with the ones reported by the authors in their papers. The convergence times of the optimized models on an nVidia GTX 1080 are 125 minutes for ResNet50 and 40 minutes for VGG16-CIFAR10.

6 Conclusion

We propose a framework to optimize NNs efficiently, it does not require a deep knowledge by the user in the optimization procedures. The framework simply takes as input a model, a dataset, and some information like batch size and the data format expected by the networks, then it returns an optimized model ready to be deployed on an edge device. The proposed framework relies on simple algorithms to solve intricate problems. The design of the framework allows future integration of new pruning and clustering algorithms keeping the original search algorithms, in the way the speedup and the compression rate can be improved. The experiments show results comparable to the state of the art on ResNet and VGG architecture.

However, the suggested solution's simplicity allows for future improvement, indeed, by using the same search algorithms, we can employ superior pruning strategies, such as filter pruning, to significantly boost inference time. The current version of the framework can handle only models for classification tasks, thus our next goal is to implement the frequently employed object detection and classification tasks. Actually, because a large number of fine-tuning phases are required, the optimizer can take time to converge, particularly on large models; therefore, it would be desirable to improve the used algorithms with more targeted solutions to reduce converge times.

Acknowledgements *This work was partially supported by the research project "EdgeVision against Varroa (EV2): Edge computing in defence of bees" funded by the Italian's MUR PRIN2022 (ERC PE6) research program.*

References

1. M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.
2. Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1398–1406, 2017.
3. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.
4. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.
5. J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," 2017.
6. S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015.
7. J. S. Larsen and L. Clemmensen, "Weight sharing and deep learning for spectral data," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4227–4231, 2020.
8. W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," 2015.
9. S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016.
10. D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, (USA), p. 1027–1035, Society for Industrial and Applied Mathematics, 2007.
11. M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021.
12. C. Liu and Q. Liu, "Improvement of pruning method for convolution neural network compression," in *Proceedings of the 2018 2nd International Conference on Deep Learning Technologies*, ICDLT '18, (New York, NY, USA), p. 57–60, Association for Computing Machinery, 2018.
13. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

14. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
15. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.
16. B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, pp. 293–299 vol.1, 1993.
17. X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," 2017.
18. W. Zeng, Y. Xiong, and R. Urtasun, "Network automatic pruning: Start nap and take a nap," 2021.
19. Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," 2019.
20. J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.
21. P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri, "Play and prune: Adaptive filter pruning for deep model compression," 2019.
22. Q. Sun, S. Cao, and Z. Chen, "Filter pruning via automatic pruning rate search," in *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pp. 4293–4309, December 2022.
23. E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "On the automatic exploration of weight sharing for deep neural network compression," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1319–1322, 2020.
24. J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, and Y. Lin, "Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions," 2018.
25. M. Gupta, E. Camci, V. R. Keneta, A. Vaidyanathan, R. Kanodia, C.-S. Foo, W. Min, and L. Jie, "Is complexity required for neural network pruning? a case study on global magnitude pruning," 2023.
26. T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," 2019.