Reviewing and discussing Graph reduction for prediction in the Edge Computing context

Asier Garmendia-Orbegozo (1), J. David Núñez-González (1), Miguel Angel Antón González (2)
(1) Department of Applied Mathematics, University of the Basque Country (UPV/EHU) Eibar, Spain
(2) TECNALIA, Basque Research and Technology Alliance (BRTA), San Sebastian, Spain

***Abstract*— Much effort has been devoted in transferring efficiently different Machine Learning (ML) algorithms and especially Deep Neural Networks (DNNs) to edge devices in order to fulfill real-time, storage and energy consumption issues, among others. Limited resources of edge devices and the necessity for energy saving to lengthen the durability of their batteries, has emerged an interesting trend on reducing neural networks and graphs while keeping almost untouched their predictability. In this work, latest works on this area are compared and analyzed in depth seeking to figure out the best techniques to reduce the dimension of those algorithms and maintain their ability at predicting. Most interesting ways of enhancing those skills are discussed, as well.**

***Keywords*— Artificial Intelligence, Edge Computing, Graph Reduction, pruning.**

I. INTRODUCTION

The use of DNNs in different scenery such as image classification, voice synthesis or object detection is undoubtedly one of the most effective ways to make predictions. The development of DNNs during the last years has evolved in such a way that nowadays neural network designs have billions of parameters with great capability of prediction, thus needing significant computation resources. Starting from huge amounts of data to be stored safely to powerful computation units, those could not be satisfied by current edge device by now. However, by reducing the size of these architectures in an efficient way it could be feasible their deployment in embedded systems.

Among others the most used and effective way to shrink these networks is the use of techniques such as pruning and quantization. The former one consists in removing parameters (neurons or weights) that have negligible contribution while maintaining the accuracy of the classifier. On the other hand, quantization involves replacing datatypes to reduced width datatypes, by transforming data to fit in new datatypes' shapes. By this way, reduced networks are able to compete with the original ones in terms of accuracy, even improving these in some cases in which overfitting issues were hindering their predictability. Moreover, by reducing the width of data edge devices could face the storage issue mentioned above and collect larger datasets in constrained memory sizes.

In this work different attempts to optimize these reduction techniques are described as well as possible future works that could be proposed to achieve even better results. The rest of this paper is organized as follows: section II introduces and analyzes the pruning process and most significant and attractive approaches made so far, section III does the same in case of quantization, and finally section IV concludes and outlines possible future research lines.

II. PRUNING

Pruning consists in removing part of connections(weights) or neurons from the original network so as to reduce the dimension of the original structure by maintaining its ability to predict. The core of this technique resides on the redundancy that some elements add to the entire architecture. Memory size and bandwidth reduction are addressed with this technique. Redundancy is lowered and overfitting is faced in some scenarios. Different classifications of works based on this ability are made depending on:

- Element pruned.
- Structured / Unstructured
- Static / Dynamic

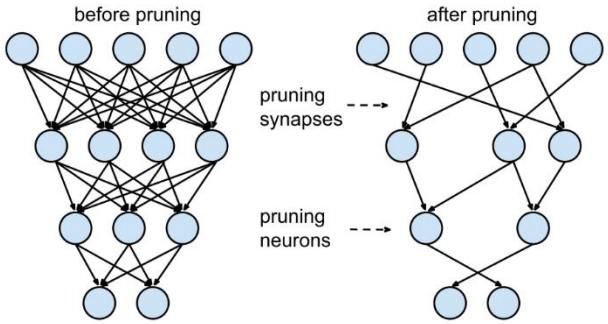

Figure 1: Different approaches for pruning. Source: https://towardsdatascience.com/ pruning-deep-neural-network-56cae1ec5505

The element pruned can be either a connection or a neuron in a pruning process. The difference between structured and unstructured pruning lies on whether the pruned network is symmetric or not. When we talk about static pruning we refer to the process in which all pruning steps are made before the inference time, while the dynamic pruning is performed during runtime.

A. Static Pruning

Static pruning is the process of removing elements of a network structure offline before training and inference processes. During these last processes no changes are made to the network previously modified. However, after removing different elements of the architecture it is interesting a fine-

tuning or retraining of the pruned network. This is due to the changes that suffer the network by removing big part of its elements. Thus, some computational effort is needed in order to reach comparable accuracy to the original network.

The pruning has been carried out by following different criteria. Some works have based on the magnitude of the elements themselves. It is undoubtedly true that near-zero values of weights make far less contribution to the results than others that surpass certain threshold value. By this way, removing connections that may appear unneeded the original network is shrunk. It is an interesting approach to develop this process layer-by-layer to not affect brutely to the performance of the resulting network. Because by removing elements of the entire network some connections or neurons may take different role in the resulting network, thus being interesting some fine-tuning or retraining.

In [1, 2] they used the second derivative of the Hessian matrix to reduce the dimension of the original architecture. Optimal Brain Damage (OBD) and Optimal Brain Surgeon (OBS) respectively function under three assumptions. Quadratic: the cost function is near quadratic. Extremal: the pruning is done after the network converged. Diagonal: sums up the error of individual weights by pruning the result of the error caused by their co-consequence. Additionally, OBS avoids the diagonal assumption and improves neuron removal precision by up to 90% reduction in weights for XOR networks. Taylor expansions of first order where also considered to reduce the network dimension in [3, 4], as a criterion to approximate the change of loss in the objective function as an effect of pruning.

Some other works have followed the idea of removing elements based on different penalization terms. Penalty-based training aims to modify or add an error function to modify weights during training process using a penalty value. At the end, near-zero values are pruned from the original network. LASSO [5] was introduced as a penalty term. It shrinks the least absolute valued feature's corresponding weights increasing weight sparsity. This operation has been shown to offer a better performance than traditional procedures such as OLS by selecting the most significantly contributed variables instead of using all the variables, achieving approximately 60% more sparsity than OLS. The problem with LASSO is that is an element-wise pruning technique leading to unstructured network and sparse weight matrices. By performing this technique group-wise as it does Group LASSO [6] removing entire groups of neurons and maintaining the original network's structure. Groups are made based on geometry, computational complexity or group sparsity among others.

Other alternatives have been proposed to carry out static pruning. In [7] was proposed a novel criterion for Convolutional Neural Network (CNN) pruning called Layer-wise relevance propagation. It is measured the contribution of each unit to the relevance of the decision making. By this way, the units that are below a predefined threshold are removed from the graph and finally the relevance of each unit is recomputed. For this last step, the total relevance per layer is calculated so that to keep it untouched during iterations. Thus, each unit's relevance is recalculated to maintain this value.

In [8] a technique to prune redundant features along with their related feature maps according to their relative cosine distances in the feature space is proposed, thus leading to smaller networks with reduced post-training inference computational costs and competitive performance. Redundancy can be reduced while inference cost (FLOPS) is reduced by 40% for VGG-16, 28%/39% for ResNet-56/110 models trained on CIFAR-10, and 28% for ResNet-34 trained on ImageNet database with minor loss of accuracy. To recover the accuracy after pruning, models were finetuned for a few iterations without the need to modify hyper-parameters.

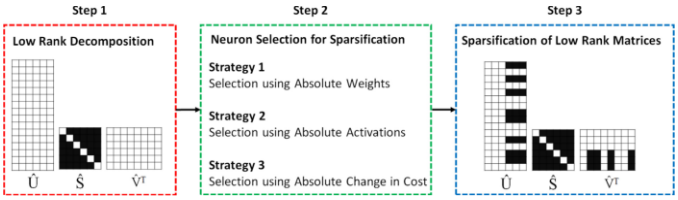

Figure 2: Architecture of the proposed approach Sparse Low Rank Decomposition in [9].

In [9] combining the ideas of sparsity and existence of un-equal contributions of neurons towards achieving the target, sparse low rank (SLR) method is presented, which sparsifies Single Value Decomposition (SVD) matrices to achieve better compression rate by keeping lower rank for unimportant neurons. By this way, it is possible to save 3.6 ×storage space of SVD without much effect on the model performance. The structured sparsity achieved by the proposed approach has also the advantage of speedup in the computation.

Another interesting approach to be taken into consideration is pruning filter-by-filter. Filter-wise pruning [10] uses the l1-norm to remove filters that do not affect the accuracy of the classification. Pruning entire filters and their related feature maps resulted in a reduced inference cost of 34% for VGG-16 model and 38% for ResNet-110 model on the CIFAR-10 dataset with improved accuracy 0.75% and 0.02%, respectively. ThiNet [11] adopts statistics information from the next layer to determine the importance of each filter. It uses a greedy search to prune the channel that has the smallest reconstruction cost in the next layer. During each training pruning is carried out more lightly to allow for coefficient stability. The pruning ratio is a predefined hyper-parameter and the runtime complexity is directly related to the pruning ratio. ThiNet compressed ResNet-50 FLOPs to 44.17% with a top-1 accuracy reduction of 1.87%.

Other research has been carried out attending activations, that may also be indicators to prune corresponding weights. Average Percentage of Zeros (APoZ) [12] was introduced to judge if one output activation map is contributing to the result. Some activation functions, particularly rectification functions such as Rectified Linear Unit (ReLU), may result in a high percentage of zeros in activations, being interesting their pruning.

After applying different techniques to reduce the amount of non-relevant elements from the original structure, it is essential a fine-tuning or retraining phase. It is shown [8] that by training a pruned structure from scratch less accurate results are obtained compared to the retraining processes in which weights from the original network are maintained for the new training phase. That is why iteratively a retraining or fine-tuning step is

followed after each pruning step is carried out. This iterative process is repeated until a desired number of elements is achieved.

B. Dynamic Pruning

Pruning a DNN dynamically offers several benefits compared to the same process carried out offline before both training and inference processes. Identifying at runtime which elements of the original structure are relevant and which ones are not, offers the possibility of solving different issues related with static pruning by adapting the network with the changes of input data.

This process is far more complex than the static one, so that various decisions are needed to make before starting it. In some cases makes sense considering additional networks or connections to further assist pruning process. Information input could be either layer-by-layer feeding a window of data iteratively to the decision system or by one-shot feeding. As well as in the static pruning, a score system and a comparative system (automatic or manual) must be established. Similarly, a stopping criterion must be imposed, and finally, the additional components have to be trained at the same time the network has been trained.

A negative impact to the system computation requirements is also needed to be taken into account. Additional bandwidth, computation and power resources are necessary while computing at runtime which elements to be pruned. At the same time, convolution operations with large number of features consume huge part of the bandwidth. Thus, a trade-off between dynamic pruning overhead, reduced network computation, and accuracy loss, should be considered. Different approaches have been developed during recent years, and the most significant ones are described below.

In [13, 14] they focused on conditional computing by activating relevant parts of the original network. The non-activated elements act as pruned ones enlightening the original structure.

The main advantage that dynamic pruning offers is the capacity of adapting the pruned network at runtime. By obtaining intermediate trained models while carrying out the whole process is an interesting way of applying a trade-off between accuracy and computation cost. In [15,16, 17] different alternatives of cascade networks were proposed. A cascade network consists of a series of networks that each of them has its output layer, instead of offering an output per-layer. Its main advantage is that it could offer an early exit if desired accuracy is achieved. On the contrary, some hyper-parameters need to be tuned manually. Moreover, in [18] Blockdrop was introduced as an Reinforcement Learning method that with an input image was able to deduce which blocks should participate in the whole process. They were able to achieve an average speed-up of 20% on ResNet-101 for ILSVRC- 2012 without accuracy loss. On the other hand, Runtime Neural Pruning (RNP) was proposed [19] based on a feature selection problem as a Markov Decision Problem (MDP) finding computation efficiency. A Recursive Neural Network (RNN) based network was used to predict which feature maps were necessary. They found 2.3x to 5.9x reduction in execution time with top-5 accuracy loss from 2.32% to 4.89% for VGG-16.

In [20] a novel dynamic pruning technique based on pruning and splicing was presented. On the one hand, pruning operations can be performed whenever the existing connections seem to become unimportant. On the other hand, the mistakenly pruned connections shall be re-established if they once appear to be important (splicing). Experimental results show that their method compressed the number of parameters in LeNet-5 and AlexNet by a factor of 108x and 17.7x, respectively, with a better learning efficiency.

The negative point of RL techniques is their computation expense. Alternatively, differentiable approaches have been made to solve this issue. Using Dynamic Channel Pruning (DCP) in [21] they proposed a side network called Feature Boosting and Suppression (FBS) to decide which channel to skip. FBS achieved 5x acceleration on VGG-16 with 0.59% ILSVRC-2012 top-5 accuracy loss, and 2x acceleration on ResNet-18 with 2.54% top-1, 1.46% top-5 accuracy loss. Similarly, in [22] a channel-threshold weighting decision (T-Weighting) was used to prune dynamically channels. A T-sigmoid activation function, using as its entry a downsampling from a Fully Connected Layer (FCL), was used to calculate channels' score and decide which ones to prune.

Another interesting approach has been proposed in [23] to prune dynamically CNNs. They explore the manifold information in the sample space to discover the relationship between different instances from two perspectives, *i.e.*, complexity and similarity, and then the relationship is preserved in the corresponding sub-networks. An adaptive penalty weight for network sparsity is developed to align the instance complexity and network complexity, while the similarity relationship is preserved by matching the similarity matrices. Extensive experiments are conducted on several benchmarks to verify the effectiveness of this method. Compared with the state-of-the-art methods, the pruned networks obtained by this method can achieve better performance with less computational cost. For example, they can reduce 55.3% FLOPs of ResNet-34 with only 0.57% top-1 accuracy drop in ImageNet.

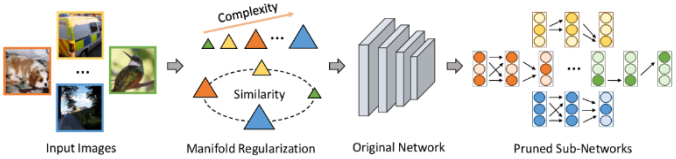

Figure 3: diagram of the architecture proposed in [23].

III. QUANTIZATION

Reducing the number of bytes used to represent data is crucial when transferring ML algorithms to resource constrained edge devices. When we mention quantization, we refer to the transformation of data from floating point representation to a determined value range. These former data could be either represented by predefined values or symbols. In the context of DNNs weights may be quantized by clustering processes, by lookup tables or using linear functions, all of them with the aim of reducing information width of data. Originally, most of the DNNs applied 32-bit floating point representation for weight parameters, but in most cases this falls excessive. In fact, 8-bit values can accelerate significantly inference process without observable loss in accuracy. Different alternatives have

been adopted in recent years, and some of the most interesting ones are described and analyzed below.

In [24] they conduct extensive experiments using incremental quantization on three applications: medical image segmentation, image classification, and automatic speech recognition. The main goal of incremental quantization is to convert 32-bit-floating-point weights (*W)* into low precision weights *W'* either power of 2 or zero with minimum accuracy loss. Each of *W'* is chosen from $Pl = \{\pm 2^{n1}, \ldots, \pm 2^{n2}\}$, here $n1$ and $n2$ are two integer numbers determined by the max absolute value of *W* of each layer and expected quantized bit-width, and $n2 \leq n1$. Experimental results in FCN for biomedical image segmentation, CNN for image classification, and RNN for automatic speech recognition, show that incremental quantization can improve accuracy by 1%, 1.95%, and 4.23% on the three applications, respectively, with 3.5x to 6.4x memory reduction.

An adaptive quantization method was introduced [25] to enhance quantization results on MNIST, CIFAR-10 and SVHN datasets, finding a unique, optimal precision for each network parameter such that the increase in loss is minimized. Pruning unnecessary parameters or quantizing them they showed DNN model sizes can be reduced significantly without loss of accuracy. The resulting models were significantly smaller than state-of-the-art quantization technique.

Following the trend of mixing pruning and quantization techniques, in [26] they presented a training acceleration framework able to speed up training process while compressing DNN for mitigating transmission overhead. FL-PSQU is a Federated Learning mechanism that is divided in three steps. First, a one-shot pruning is done by the server to generate general models for all clients and after quantizing it, it is transferred to each client. Then, each client updates its model and depending on its update's significance it is transmitted to the server or not, avoiding unnecessary communications by this way.

Other simpler approaches have been made that include binary and ternary quantization. Thus, only two or three possible values were assigned to each element of the architecture, with a vast reduction in memory size and computation effort. However, accuracy loss is not negligible in these techniques due to the hard generalization of weights. Binary Connect [27] used stochastic plus-minus quantization by assigning +1 to positive valued weights and -1 to negative valued ones with hard sigmoid probability $\sigma(x)$ and $1- \sigma(x)$, respectively. Similarly, in [28] they binarized all weights of different architectures and afterward multiply with a scaling factor for all the weights of a layer. In [29, 30] showed the power of ternary quantization by including additional value (-1, 0 and +1) compared to the binary case. Its implementation in hardware must be efficient due to the fact that 0 value does not actually participate in computations. Ternary Weight Networks (TWN) adopts the l2-distance to find the scale and formats the weights into -1, 0 and +1 with a threshold generated by an assumption that the weighs are uniformly distributed such as in [-a, a]. This resulted in up to 16x model compression with 3.6% ResNet-18 top-1 accuracy loss on ILSVRC-2012.

In some scenarios could be of special interest the conversion of floating-point multiplication to bit-shift operations, such as in case of FPGAs. Here, the constraint for weights for being power of two, leverages training and inference effort and time. This approach was proposed in [31] by quantizing the representation of weights layer-by-layer. Likewise, Incremental Network Quantization [32] replaces all weights with powers of 2 iteratively, preserving in each iteration some weights in full precision and retraining them. After multiple iterations majority of the weights are converted to power-of-two. The final structure has weights from 2 to 5 bits with values near zero set to zero. Results of group-wise iterative quantization show lower error rates than a random power-of-two strategy.

IV. CONCLUSION AND FUTURE WORK

Among different approaches that have been made so far in the literature to reduce the dimension of the original network size and optimize inference time, memory usage and computational cost, the ones mentioned above are the most interesting ones from our point of view. When transferring deep networks usually trained on cloud to edge devices, it is of great interest a reduction in network size to adapt these architectures to the constraints of such devices. Thus, pruning original network is undoubtedly an essential step to fit deep models in resource constrained devices. Dynamic pruning offers the possibility of removing unnecessary elements of the network at inference time, thus, offering way of achieving the most accurate reduction for the input data. In fact, these techniques are computationally more expensive than the static ones, and quite a bit complex. For some applications some approaches will suit better than others and it is up to the final user the election among them. However, achieving a computationally effortless way of pruning networks dynamically would be an interesting future research line.

On the other hand, quantization techniques would not be underestimated specially when transferring models to microcontrollers that lack of operating systems. Specially when those microcontrollers own FPGAs, a conversion from floating-point multiplications to bit-shift operation could alleviate a great computational cost and reduce runtime, as well. In the majority of scenarios reducing the number of representative bits to 8 is enough for avoiding a significant loss in terms of accuracy while reducing vastly memory usage. Nonetheless, some other approaches like binary quantization are too hard in many cases leading to a excessive accuracy drop.

REFERENCES

[1] Y. LeCun, J.S. Denker, S.A. Solla, Optimal Brain Damage, Advances in Neural Information Processing Systems (NIPS) (1990) *598–605*, https://doi.org/ 10.5555/109230.109298.

[2] Hassibi, B., et al. Optimal brain surgeon and general network pruning. 10.1109/icnn.1993.298572. 1993.

[3] P. Molchanov et al. Pruning convolutional neural networks for resource efficient transfer learning. Proceedings of International Conference on Learning Representations (ICLR) 2017.

[4] C. Yu et al. Transfer channel pruning for compressing Deep domain adaptation models. Int. J. Mach. Learn. Cybern, 10 (11) (2019) *3129-3144.*

[5] R. Muthukrishnan, R. Rohini. LASSO: A feature selection technique in predictive modeling for machine learning, in: 2016 IEEE International Conference on Advances in Computer Applications (ICACA), IEEE, 2016, pp. *18–20*, https://doi.org/10.1109/ICACA.2016.7887916.

[6] M. Yuan, Y. Lin. Model selection and estimation in regression with grouped variables, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68 (2006) *49–67*, https://doi.org/10.1111/j.1467-9868.2005.00532.x, http://doi.wiley.com/10.1111/j.1467-9868.2005.00532.x,.
[7] S.K. Yeom et al. Pruning by explaining: A novel criterion for Deep neural network pruning. Pattern Recognition 115 (2021) 107899.
[8] B. O. Ayinde et al. Redundant feature pruning for acelerated inference in deep neural neutworks. Neural Networks 118 (2019) *148-158.*
[9] S. Swaminathan et al. Sparse low rank factorization for deep neural network compression. Neurocomputing 398 (2020) *185-196*.
[10] H. Li et al. Pruning Filters for Efficient ConvNets, in: International Conference on Learning Representations (ICLR), 2017, https://doi.org/10.1029/2009GL038531.
[11] J.H.H. Luo, J. Wu, W. Lin. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2017-Octob, *5068–5076*, https://doi.org/10.1109/ICCV.2017.541.
[12] H. Hu et al. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. 2016. ArXiv preprint http://arxiv.org/abs/1607.03250. .
[13] Y. Bengio. Estimating or Propagating Gradients Through Stochastic Neurons. 2013. ArXiv preprint http://arxiv.org/abs/1305.2982. .
[14] A. Davis, I. Arel. Low-Rank Approximations for Conditional Feedforward Computation in Deep Neural Networks, International Conference on Learning Representations.
[15] S. Leroux et al. The cascading neural network: building the Internet of Smart Things. Knowl. Inf. Syst. 52 (2017) *791–814,* https://doi.org/10.1007/s10115-017-1029-1, http://link.springer.com/10.1007/s10115-017-1029-1.
[16] T. Bolukbasi et al. Neural Networks for Efficient Inference, in: Thirty-fourth International Conference on Machine Learning, 2017.
[17] A. Odena et al. Changing Model Behavior at Test-Time Using Reinforcement Learning, in: International Conference on Learning Representations Workshops (ICLRW), International Conference on Learning Representations, ICLR. 2017. http://arxiv.org/abs/1702.07780. .
[18] Z. Wu et al. BlockDrop: Dynamic Inference Paths in Residual Networks, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2018, pp. *8817–8826*, https://doi.org/10.1109/CVPR.2018.00919.
[19] J. Lin et al. Runtime Neural Pruning. Advances in Neural Information Processing Systems (NIPS). 2017. *2178–2188*, https://papers. nips.cc/paper/6813-runtime-neural-pruning.pdf.
[20] Y. Guo et al. Dynamic Network Surgery for Efficient DNNs. 30th Conference on Neural Information Processing Systems (NIPS). 2016. Barcelona, Spain.
[21] X. Gao et al. Dynamic Channel Pruning: Feature Boosting and Suppression, in: International Conference on Learning Representations (ICLR), 2019, pp. *1–14*, http://arxiv.org/abs/1810.05331.
[22] Z. Chiliang et al. Accelerating Convolutional Neural Networks with Dynamic Channel Pruning, in: 2019 Data Compression Conference (DCC), IEEE, 2019, p. *563*, https://doi.org/10.1109/DCC.2019.00075.
[23] Y. Tang et al. Manifold Regularized Dynamic Network Pruning. Computer Vision Foundation. 2021.
[24] W. Chen et al. Quantization of Deep Neural Networks for Accurate Edge Computing. ACM Journal on Emerging Technologies in Computing Systems, Vol. 17, No. 4, Article 54. 2021. https://doi.org/10.1145/3451211
[25] S. Khoram and J. Li. Adaptive Quantization of Neural Networks, in: ICLR 2018.
[26] W. Xu et al. Accelerating Federated Learning for IoT in Big Data Analytics With Pruning, Quantization and Selective Updating. IEEE Access Vol. 9, 2021. *38457-38466.*
[27] M. Courbariaux, et al. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. 2016. ArXiv preprint https://github.com/MatthieuCourbariaux/ http://arxiv.org/abs/1602.02830.
[28] M. Rastegari et al. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, in: European Conference on Computer Vision, Springer. 2016. pp. *525–542*. http://arxiv.org/abs/1603.05279 http://link.springer.com/10.1007/978-3-319-46493-0_32, DOI: 10.1007/978-3-319-46493-0_32. .
[29] F. Li et al. Ternary Weight Networks, in: Advances in Neural Information Processing Systems (NIPS). 2016. http://arxiv.org/abs/1605.04711.
[30] C. Leng et al. Extremely Low Bit Neural Network: Squeeze the Last Bit Out with ADMM, in: The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018.
[31] Z. Lin et al. Neural Networks with Few Multiplications, in: International Conference on Learning Representations (ICLR), 2016.
[32] A. Zhou et al. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights, in: International Conference on Learning Representations (ICLR), 2017.