An Efficient Approach to Iterative Network Pruning

Chuan-Shun Huang[1], Wuqian Tang[1], Yung-Chih Chen[2], Yi-Ting Li[1], Shih-Chieh Chang[1], and Chun-Yao Wang[1]

[1]National Tsing Hua University, Taiwan, ROC

[2]National Taiwan University of Science and Technology, Taiwan, R.O.C.

2024 International VLSI Symposium on Technology, Systems and Applications (VLSI TSA) | 979-8-3503-6034-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/VLSITSA60681.2024.10546442

Abstract—**Network pruning is a technique to minimize the number of parameters of large neural networks. Network pruning can be performed once or multiple times. One-shot network pruning is easy to reach the required sparsity, but the corresponding accuracy drop may be unacceptable with respect to different goals. On the other hand, iterative network pruning trims and retrains the network iteratively to maintain the accuracy, but suffering from the long runtime of this repetitive procedure. In this work, we propose an efficient approach to network pruning by removing redundant trainings. Experimental results show that our approach reduces 25% to almost 60% of training time with comparable network accuracy as compared to the state-of-the-art.**

I. Introduction

The progression of cutting-edge technologies like Quantum Computing [18], AI [3], [11], [29], and Approximate Computing [14], [16], [27] signals a paradigm shift in our handling of complex computational tasks. Deep Neural Networks (DNNs) related applications in AI have become increasingly popular. However, the large size of DNN is still a challenge for engineers when deploying DNNs on devices. A solution to reducing the size of DNNs is network pruning [7]–[9], whose goal is to eliminate components or weights from neural networks while preserving the accuracy. To recover from the possible accuracy drop after pruning, retraining plays an important role. Fine-tuning (FT) [8] is a conventional method by retraining the pruned network with the minimum learning rate used in the training schedule. Learning rate rewinding (LRW) [23] adopts the training schedule of the original network based on the last few epochs, and shows a better accuracy than fine-tuning. However, recent research [15] shows that by using cyclical learning rate (CLR) [25], [26], it outperforms previous techniques with a much better accuracy recovery.

Unfortunately, retraining techniques still suffer from a long runtime when performing iterative pruning. This technique is capable of preserving the accuracy, but performing the retraining process iteratively is very time-consuming. One-shot pruning [8] is a faster method to perform network pruning by setting the target sparsity and retraining once. However, the accuracy loss might be a serious concern when setting to a high sparsity.

[4] observed that pruning at most 20% weights from the network per iteration balances efficiency with network accuracy. This method is still used by recent research [15], [23] that seeks for better retraining techniques. However, as better retraining techniques proposed, [4] can be further improved since this method was not combined with those better retraining techniques. By comparing the accuracy of pruned networks retrained with FT [8] and CLR [26] separately, we found that the 20% pruning ratio limit [4] is unnecessary when pruning at a low network sparsity. We also observed that better retraining techniques can further extend the pruning ratio in the first pruning iteration.

In this work, our pruning strategy focuses on unstructured pruning [8], which is a network pruning technique that trims connections between neurons only. To save time from the traditional iterative pruning, our method reduces retraining time by extending the pruning ratio of the first pruning iteration. Our experimental results show that network pruning iterations of [4] at low sparsity can be combined into one step. After the first iteration, we at most need two more iterations with small pruning ratio to complete the process, where [4] usually needs 4 or 5 iterations to reach high sparsity. Our method costs 25% to 60% time and is still able to get comparable results as compared to [4].

II. Preliminaries

A. Network Pruning

Network pruning can be classified into two categories – unstructured pruning and structured pruning. Unstructured pruning [2], [8] removes the less important connections in the network, which is usually performed by changing the weights to zero. Hence, it is also called weight pruning. The result of unstructured pruning is a network with more sparse matrices. Although the size of pruned network does not necessarily become smaller, its computation can be accelerated by hardware that supports sparse matrix computation. Structured pruning [17], [30], on the other hand, removes less important neurons in the network. This is usually performed by pruning weights in groups, such as components like channels, filters, or layers. By removing the components, the pruned network has fewer parameters than the original network. However, pruning neurons might remove both important and unimportant connections at the same time, which may lead to more accuracy loss [6].

This work is supported in part by the National Science and Technology Council (Taiwan) under MOST 109-2221-E-007-082-MY2, MOST 111-2221-E-007-121, MOST 111-2221-E-011-137-MY3, NSTC 112-2218-E-007-014, NSTC 112-2221-E-007-106-MY2, NSTC 112-2221-E-007-108, NSTC 112-2425-H-007-002, and NSTC 113-2425-H-007-004.

B. Magnitude-Based Weight Pruning

Magnitude-based Weight Pruning (MWP) is an efficient unstructured pruning algorithm proposed by [8]. MWP compares the absolute values of the weights, and considers the smaller value as "carrying less information". Weights with smaller magnitudes would be removed, or change into zero. MWP removes these weights in an ascending order in the network. In this work, we globally consider the absolute values of all trainable weights across the whole network. However, for ResNets or networks with a single fully-connected layer, we perform MWP only on convolution layers.

C. Retraining Techniques

After pruning a network, its accuracy may decrease due to parameter modification. To change the weights that fits the pruned network, there are several techniques used to retrain the networks for regaining accuracy.

1. Fine-Tuning: Fine-tuning (FT) [8] is a retraining method that sets a small constant learning rate to retrain the pruned network. The learning rate that FT uses is based on the learning rate schedule for training the initial network. To train an initial network, a conventional method is to train with a multi-step learning rate schedule. These schedules usually start with a large learning rate α, e.g., 0.1, and use it to train the initial network for several epochs. Then, a smaller learning rate would be applied, usually one-tenth of the previous one, for few more epochs. At last, the smallest learning rate is used to train the network. An example of this kind of learning rate schedule is listed in TABLE III for training ResNets [10]. To fine-tune ResNet, we set α = 0.001, as our constant FT learning rate.

2. Cyclical Learning Rate: Cyclical Learning Rate (CLR) [25], [26] is a learning rate schedule that converges networks rapidly. CLR's schedule can be divided into two parts: rising and falling. The rising part starts with a small learning rate and increases to a prespecified learning rate in few epochs. This process is also known as warm up. The falling part starts from the prespecified learning rate, and decreases with a cosine function through several epochs. The rising and falling parts can be performed alternately for many cycles. However, [15] uses 1-cycle CLR to retrain pruned networks. [15] observed that 1-cycle CLR converges the network faster and gets a better accuracy than FT.

D. One-Shot Pruning and Iterative Pruning

A standard network pruning process starts with a well-trained original neural network. Next, it applies a pruning algorithm, e.g., MWP, on this network to reach a target sparsity. Finally, it retrains the pruned network with a retraining technique for several epochs to regain the accuracy loss.

For one-shot pruning [8], [17], network pruning and retraining are both conducted once. That is, pruning the network to the required sparsity in one move, then retrain the model only once. The advantage of one-shot pruning is time saving. However, when we aim for a high network sparsity, the accuracy is usually unrecoverable due to the massive changes of network parameters. The sparsity that the one-shot pruning is able to

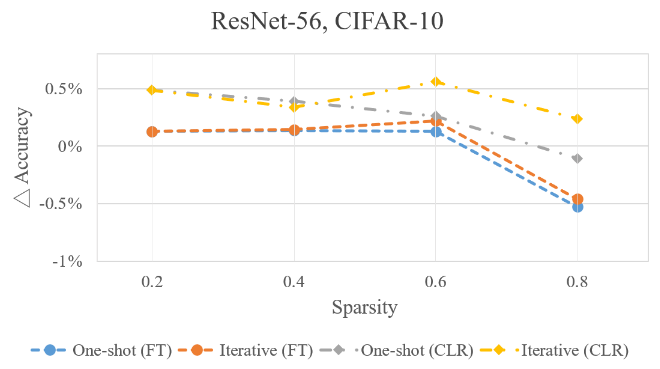

Fig. 1: One-shot and iterative pruning results of ResNet-56 on CIFAR-10 using MWP [8] and retrained by FT/CLR for 20 epochs, respectively. The retraining is applied after each iteration in the iterative pruning.

remain acceptable accuracy depends on the network structure, pruning algorithm, and retraining techniques. For instance, a pruned network can reach a higher sparsity with one-shot pruning when it is retrained with CLR rather than with FT. However, even with a decent pruning or retraining technique, one-shot pruning would fail and cause serious accuracy loss when targeting at a certain high level of sparsity.

Iterative pruning [8], [17] repeats the process of pruning and retraining for multiple times until the network reaches the target sparsity. By removing a smaller percentage of weights one time, it is easier to retrain the pruned network and regain accuracy. With the iterative pruning, it is much easier to remain network accuracy at a high sparsity, but the process is very time-consuming.

III. The Proposed Approach

The idea of iterative pruning is to prune a portion of parameters, retrain the pruned network to regain accuracy such that it is able to withstand accuracy loss in the next iteration. [4] prunes 20% of weights in the network per iteration until reaching a required sparsity, and this method was also adopted in several studies [15], [19], [23]. However, we found that this process can be accelerated by digging into the potential of the 1-cycle CLR [15], [26].

A. *High Pruning Ratio for the First Iteration*

When pruning at a low sparsity, the accuracies after retraining are similar for both one-shot and iterative pruning, e.g., for ResNet-56 [10] on CIFAR-10 [13] as shown in Fig 1. The accuracy is almost the same for one-shot and iterative pruning under 40% sparsity. If we aim for a 40% sparsity network, using one-shot pruning costs only half time of iterative pruning while having a good outcome. This phenomenon inspires us to use a larger prune ratio at the first iteration of iterative pruning.

To verify if using a larger pruning ratio for the first iteration is possible, we set our target sparsity as 80%, where the work [4] would take 4 iterations of pruning and retraining. We increase the prune ratio to 40% in the first iteration, and the result is shown in TABLE I. Pruning 40% weights in the first move has a similar network accuracy compared to pruning 20% weights evenly in all iterations, but only requires 75% training epochs.

According to Fig 1, we can also notice one-shot pruning with CLR outperforms both methods that retrains with FT, and the ability of CLR to recover the accuracy at high network sparsity is remarkable. By applying more retraining efforts for high sparsity network pruning, we can further elevate the prune ratio at the first iteration and still able to remain comparable performance. As shown in TABLE I, if we apply 10 more epochs for retraining at each iteration, we can set a 60% pruning ratio in the first iteration and requires 75% training epochs.

Previous experimental results lead to our method: For a given target network sparsity $n\%$, we set our first iteration pruning ratio $m\%$. We finish the network pruning process in the next iteration, where we prune the retrained network once more to reach $n\%$ sparsity. Thus, we can complete iterative pruning in 2 iterations.

We set the value of m as a smaller number than n, where $5 \leq n - m \leq 10$. We do not want $n - m$, or the pruning ratio increased for the second iteration pruning to be too large; otherwise, it might lead to an unrecoverable accuracy drop when pruning to a high target sparsity. In our experiment, we set the value of m as the largest multiples of ten or five that satisfies the mentioned formula.

B. *Take an Extra Step Back to a Resilient Point*

Over-parameterized neural networks are often easier to retrain than networks with fewer parameters [21] after pruning. These large networks also have more tolerance when pruning a large portion of weights in the first iteration. When aiming for a high sparsity with our method, the pruning ratio $m\%$ in the first iteration can be set as 80% for large networks like VGG-nets [24]. However, networks with much fewer parameters like ResNet-56 on CIFAR-10, as shown in Fig 1, 80% sparsity is about the limit for CLR to recover the accuracy. By using more retraining time, we can regain accuracy to the original one at 80% sparsity, but not able to push the accuracy even higher to endure further network pruning.

When handling these small models, we provide one extra step back: If the first pruning ratio $m\%$ cannot get a decent accuracy after retraining with CLR, we set a pruning ratio $k\%$, which replaces $m\%$ as the first iteration pruning ratio. The $k\%$ stands for a $resilient\ point$, where it becomes much more difficult to regain network accuracy if we set a sparsity number larger than k to perform the first pruning move.

Our method then becomes a 3-iteration network pruning: pruning to $k\%$ sparsity, or the $resilient\ point$ first, $m\%$ sparsity afterwards, and finally to the target sparsity $n\%$. For larger models, the value of k is usually the same as m, which is still considered as a 2-iteration network pruning.

From the experimental results of [4], setting the pruning ratio per iteration smaller preserves the accuracy better. However, if the pruning ratio chosen is too small, the whole iterative pruning process then becomes extremely time consuming. Thus, we search the k value by subtracting 10 from m repeatedly to lower our search time. In our experiment, we can usually find the $resilient\ point$ in the first search, and we at most searched two times for the ResNet-110 on CIFAR-10 case.

Similar concepts have also been mentioned in [17] when pruning filters across different layers. [17] shows that layers that are insensitive to network pruning can be pruned by one-shot and still able to regain accuracy easily. However, sensitive layers should be pruned iteratively; otherwise the accuracy drop would be unrecoverable. Applying this concept to our situation, a network remains resilient to network pruning before it reaches the $resilient\ point$, or $k\%$ sparsity, where we are able to retrain smoothly; but pruning more weights will cause the accuracy drop visibly. Hence, we need to take smaller pruning steps afterwards, like pruning with a small portion to $m\%$ and $n\%$.

As shown in the row 2 of TABLE II, we set m = 80%, n = 90%, but not able to recover the accuracy drop completely since pruning 80% in the first move is too much for CLR to retrain well. We set k = 70%, however, and perform a 3-iteration network pruning: 70%, 80%, and 90%, we are able to reduce 25% of retraining time and get a comparable result to [4].

IV. Experimental Results

All experiments were conducted on Windows 10, with Intel i5-13600 CPU, Nvidia RTX4090 GPU, and 64GB RAM. We use the following networks in our experiments: ResNet-56 [10], ResNet-110 [10] , VGG-16 [24], and R(2+1)D-18 [28]. The training configurations of these networks are listed in TABLE III. All experiments are run for three times, and the accuracy is reported as "$mean{\pm}std$".

A. *ResNets on CIFAR-10*

For ResNets [10], we choose ResNet-56 and ResNet-110, and use CIFAR-10 as our dataset. We used the train code of [12], and adopted the hyperparameters from [15] to train our initial network. We use global MWP [8] to prune all convolution layers, and the small fully-connected layer at the end of the network remains unpruned [19]. We set our target sparsity as 90%, which is difficult for CLR to recover the accuracy with one-shot pruning, that means iterative pruning is the only option here.

To retrain the network, we use the 1-cycle CLR [15], [26]. We take 3 warmup epochs to reach the largest learning rate used in the original training schedule. For the rest of the retraining epochs, we use cosine function to decrease the learning rate.

As the results shown in TABLE IV, we provide the 3-iteration pruning for both ResNet-56 and ResNet-110 due to their small network sizes, which made both networks very sensitive at a high sparsity. For ResNet-56, we set our $resilient\ point$ at 70% sparsity; for ResNet-110, we set 60% sparsity as a better $resilient\ point$.

Our results for ResNet-56 is comparable to [4]; but the results of ResNet-110 is better than [4]. Since networks with fewer parameters are harder to retrain, the reduction of time is only 25% as compared to [4].

TABLE I: Results of iterative pruning for ResNet-56 on CIFAR-10 with different prune rates for the first iteration.

Network	Acc Ori.(%)	Pruning Schedule(%)	Accuracy(%)	Epochs/Iter	Total Epochs	Epochs↓(%)
ResNet-56	93.22	20, 40, 60, 80	93.46 ± 0.13	20	80	0
		40, 60, 80	93.47 ± 0.15	20	60	25
		60, 80	93.68 ± 0.17	30	60	25

TABLE II: Results of iterative pruning for ResNet-56 on CIFAR-10 when aiming for 90% sparsity.

Network	Acc Ori.(%)	Pruning Schedule(%)	Accuracy(%)	Epochs/Iter	Total Epochs	Epochs↓(%)
ResNet-56	93.22	20, 40, 60, 80, 90	93.31 ± 0.09	40	200	0
		80, 90	93.21 ± 0.03	80	160	20
		70, 80, 90	93.30 ± 0.18	50	150	25

TABLE III: Training configurations for the unpruned networks in the experiments.

Dataset	Network	|Param|	Optimizer	Learning Rate (t = training epoch)
CIFAR-10	ResNet-56	0.85M	Nesterov SGD β = 0.9 Batch size: 64 Weight decay: 0.0001 Epochs: 160	$\alpha = \begin{cases} 0.1 & t \in [0, 80) \\ 0.01 & t \in [80, 120) \\ 0.001 & t \in [120, 160] \end{cases}$
CIFAR-100	ResNet-110	1.72M		
CIFAR-10	VGG-16	15.2M	Nesterov SGD β = 0.9 Batch size: 128 Weight decay: 0.0005 Epochs: 300	$\alpha = \begin{cases} 0.05 \cdot \frac{1}{2}^{\lfloor \frac{t}{30} \rfloor} & t \in [0, 300] \end{cases}$
EgoGesture	R(2+1)D-18	31.3M	Adam β_1 = 0.9 β_2 = 0.999 Batch size: 16 Epochs: 50	$\alpha = \begin{cases} \frac{0.001}{2} \cdot (1 + \cos(\frac{t \bmod 10}{9})\pi) & t \in [0, 50] \end{cases}$

TABLE IV: Results of iterative pruning with [4] comparing to our method when setting a high sparsity for ResNet-56, ResNet-110, VGG-16 on CIFAR-10. We prune all the networks with MWP [8] globally, and retrained with CLR after each pruning iteration.

Network	Acc Ori.(%)	Param↓(%)	Method	Accuracy(%)	Epochs/Iter	Total Epochs	CPU(s)	Time↓(%)
ResNet-56	93.22	90	[4]	93.31 ± 0.09	40	200	4046	0
			Ours	93.30 ± 0.18	50	150	3030	25.11
ResNet-110	93.50	90	[4]	93.57 ± 0.24	40	200	5561	0
			Ours	93.66 ± 0.08	50	150	4155	25.28
VGG-16	92.50	96	[4]	92.56 ± 0.12	40	200	3036	0
			Ours	92.58 ± 0.20	40	80	1240	59.16

B. VGG-16 on CIFAR-10

For VGG-16 [24], we also set CIFAR-10 as the dataset, and adopt the training configurations from [5] to train our initial network. We set the initial learning rate as 0.05, and we divide it by 2 for every 30 epochs. The training process lasts for 300 epochs.

Since VGG-16 has 3 large fully-connected layers, we apply global MWP [8] to prune all layers. We set the target sparsity as 96% due to its high tolerance to network pruning. 96% sparsity is also a target that (one-shot MWP + CLR) cannot recover the network's accuracy.

For retraining, we modified our initial learning rate to 0.01. It has a better retraining performance compared to the initial learning rate 0.05 that we used in the initial network training schedule. 3 warmup epochs and the cosine decay function for the rest of the retraining is the same experimental settings as for ResNets.

As shown in TABLE IV, we provide the 2-iteration pruning for VGG-16. We set our $resilient\ point$ at 90%, and move on to 96% for the next iteration. Compared to [4], which spends 5 iterations to reach 96% sparsity, we only need 2 iterations and save 59% CPU time with a slightly better accuracy.

C. ResNets on CIFAR-100

We conduct our experiments for ResNets on CIFAR-100[13], to showcase a situation that is hard to preserve the network accuracy. We use the same experimental settings as for ResNets on CIFAR-10.

As shown in TABLE V, we cannot achieve accuracy no drop with either methods when aiming for a 90% network sparsity. However, the results are very similar to the experiments for ResNets shown in TABLE IV. We also get a comparable accuracy for ResNet-56, and a slightly better accuracy for ResNet-110 when compared to [4], with about 75% runtime.

D. R(2+1)D-18 on EgoGesture

We selected R(2+1)D-18 [28] for our experiment. We would like to show the performance of the network pruning and retraining techniques when applying on a very different dataset and network structure.

We adopted the pre-trained parameters of R(2+1)D-18 from [22] and continued to train with our learning rate schedule and the selected datasets. We select EgoGesture [1], [31] as our dataset. EgoGesture contains 83 kinds of static or dynamic hand gesture video clips. We also picked 10 classes of gestures to form an "EgoGesture-10cls" dataset, in order to simulate a 10-classification dataset like CIFAR-10. To be more specific, we picked hand gestures that are labeled as number 1, 2, 3, 4, 5, 6, 12, 13, 14, and 15, which are all dynamic hand gestures.

We set CosineAnnealingWarmRestarts [20] as our learning rate scheduler, where it decreases the learning rate from 0.01 to 0 in 10 epochs, and restarts at the initial learning rate again. We use 50 epochs to train the network on the both EgoGesture-10cls and full EgoGesture dataset.

Similar to ResNets, we also exclude R(2+1)D-18's fully-connected layer when performing global MWP [8]. When retraining, we do not apply any warmups since it has a rather small initial learning rate.

The experimental results for R(2+1)D-18 on EgoGesture-10cls dataset are shown in TABLE VI. We set our $resilient\ point$ at 80% and perform 2-iteration pruning. We are managed to use almost 40% time to achieve accuracy no drop while [4] failed to do so.

TABLE V: We apply the same experimental settings in TABLE IV for ResNet-56 and ResNet-110 on CIFAR-100.

Network	Acc Ori.(%)	Param↓(%)	Method	Accuracy(%)	Epochs/Iter	Total Epochs	Epochs↓(%)
ResNet-56	69.66	90	[4]	66.65 ± 0.08	40	200	0
			Ours	66.64 ± 0.16	50	150	25
ResNet-110	71.49	90	[4]	68.73 ± 0.31	40	200	0
			Ours	68.79 ± 0.12	50	150	25

TABLE VI: Results of iterative pruning with [4] comparing to our method when setting 90% sparsity for R(2+1)D-18 on EgoGesture-10cls and full EgoGesture (83cls) dataset.

Dataset	Acc Ori.(%)	Param↓(%)	Method	Accuracy(%)	Epochs/Iter	Total Epochs	CPU(s)	Time↓(%)
EgoGesture-10cls	96.85	90	[4]	96.68 ± 0.72	20	100	4518	0
			Ours	96.85 ± 0.33	20	40	1809	59.96
EgoGesture-83cls	91.76	90	[4]	89.41 ± 0.45	20	100	50202	0
			Ours	89.49 ± 0.45	30	60	30753	38.74

The experimental results for R(2+1)D-18 on EgoGesture full dataset are also shown in TABLE VI. We observed the same phenomenon as shown in the CIFAR-100 experiment - accuracy is hard to preserve for datasets that contain more classes. By adding 10 extra retraining epochs for each iteration, we get a slightly better result than [4] with almost 61% runtime.

V. Conclusion

In this work, we propose an efficient approach to network pruning with comparable network accuracy. We use a large network pruning portion in the first pruning iteration to lower the runtime of iterative pruning process. Furthermore, our results on accuracy are able to match the state-of-the-art iterative pruning method on various network structures and datasets with less runtime.

References

[1] C. Cao, Y. Zhang, Y. Wu, H. Lu, and J. Cheng, "Egocentric gesture recognition using recurrent 3d convolutional neural networks with spatiotemporal transformer modules," in *Proc. of the IEEE International Conference on Computer Vision*, 2017, pp. 3763–3771.

[2] M. A. Carreira-Perpinán and Y. Idelbayev, ""Learning-compression" algorithms for neural net pruning," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8532–8541.

[3] Y.-C. Chang, C.-C. Lin, Y.-T. Lin, Y.-C. Chen, and C.-Y. Wang, "A convolutional result sharing approach for binarized neural network inference," in *Proc. of the IEEE Design, Automation & Test in Europe Conference & Exhibition*, 2020, pp. 780–785.

[4] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. of the International Conference on Learning Representations*, 2019.

[5] C.-Y. Fu, *PyTorch-VGG-CIFAR10*, https://github.com/chengyangfu/pytorch-vgg-cifar10, Accessed: 2023-06-23.

[6] K. Ghodasara, "Overview of decision tree pruning in machine learning," *International Research Journal of Engineering and Technology*, vol. 8, no. 8, pp. 2073–2076, 2021.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *Proc. of the International Conference on Learning Representations*, 2016.

[8] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[9] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in Neural Information Processing Systems*, vol. 5, 1992.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[11] Y.-C. Huang, Y.-H. Tsai, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "Accelerating binarized neural network inference by reusing operation results and elevating resource utilization on edge devices," in *Proc. of the IEEE International VLSI Symposium on Technology, Systems and Applications*, 2023, pp. 1–4.

[12] Y. Idelbayev, *Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch*, https://github.com/akamaster/pytorch_resnet_cifar10, Accessed: 2023-05-14.

[13] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[14] Y.-A. Lai, C.-C. Lin, C.-C. Wu, Y.-C. Chen, and C.-Y. Wang, "Efficient synthesis of approximate threshold logic circuits with an error rate guarantee," in *Proc. of the IEEE Design, Automation & Test in Europe Conference & Exhibition*, 2018, pp. 773–778.

[15] D. H. Le and B. Hua, "Network pruning that matters: A case study on retraining variants," in *Proc. of the International Conference on Learning Representations*, 2021.

[16] C.-T. Lee, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "Approximate logic synthesis by genetic algorithm with an error rate guarantee," in *Proc. of the IEEE Asia and South Pacific Design Automation Conference*, 2023, pp. 146–151.

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. of the International Conference on Learning Representations*, 2017.

[18] H.-L. Liu, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "A robust approach to detecting non-equivalent quantum circuits using specially designed stimuli," in *Proc. of the IEEE Asia and South Pacific Design Automation Conference*, 2023, pp. 696–701.

[19] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *Proc. of the International Conference on Learning Representations*, 2019.

[20] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *Proc. of the International Conference on Learning Representations*, 2017.

[21] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "The role of over-parametrization in generalization of neural networks," in *Proc. of the International Conference on Learning Representations*, 2019.

[22] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[23] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," in *Proc. of the International Conference on Learning Representations*, 2020.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of the International Conference on Learning Representations*, 2015.

[25] L. N. Smith, "Cyclical learning rates for training neural networks," in *IEEE Winter Conference on Applications of Computer Vision*, 2017, pp. 464–472.

[26] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Proc. of Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, SPIE, vol. 11006, 2019, pp. 369–386.

[27] K. S. Tam, C.-C. Lin, Y.-C. Chen, and C.-Y. Wang, "An efficient approximate node merging with an error rate guarantee," in *Proc. of the IEEE Asia and South Pacific Design Automation Conference*, 2021, pp. 266–271.

[28] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6450–6459.

[29] Y.-H. Tsai, Y.-C. Huang, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "Minimizing computation in binarized neural network inference using partial-filter sharing," in *Proc. of the IEEE International VLSI Symposium on Technology, Systems and Applications*, 2023, pp. 1–4.

[30] R. Yu, A. Li, C.-F. Chen, *et al.*, "Nisp: Pruning networks using neuron importance score propagation," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[31] Y. Zhang, C. Cao, J. Cheng, and H. Lu, "Egogesture: A new dataset and benchmark for egocentric hand gesture recognition," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1038–1050, 2018.