2022 26th International Conference on Pattern Recognition (ICPR)
August 21-25, 2022, Montréal, Québec, Canada

Genetic-based Joint Dynamic Pruning and Learning Algorithm to Boost DNN Performance

Azadeh Famili
Holcombe Department of
Electrical and Computer Engineering
Clemson University
Clemson, USA
Email: agholam@clemson.edu

Yingjie Lao
Holcombe Department of
Electrical and Computer Engineering
Clemson University
Clemson, USA
Email: ylao@clemson.edu

2022 26th International Conference on Pattern Recognition (ICPR) | 978-1-6654-9062-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICPR56361.2022.9956310

***Abstract*—The learning process of a biological system is a continuous phenomenon with limited external interventions. As learning progress, the numbers of neurons and synapses are modified based on the circumstances, which will impact the learning rate (i.e., learning faster as learning progresses). However, different from the characteristics of biological systems, the current research on deep learning is focused on a fixed training process with a predefined architecture to obtain optimal accuracy. On the other hand, while model pruning techniques have been studied to eliminate redundant neurons or synapses, most of them are applied after training but before deployment to accelerate the inference. In this paper, we integrate pruning into training and propose a genetic-based joint pruning and learning algorithm that monitors the training process and prunes the redundant parameters while training. As a result, our method can accelerate both training and inference. The proposed genetic-based method is well-suited for both training from scratch and online learning tasks by considering both the importance and stability of the parameters in the pruning process. The effectiveness of the proposed algorithm is evaluated on different neural network architectures and datasets, which demonstrates significant improvements for the training under both batch learning and incremental learning scenarios.**

I. Introduction

The original inspiration behind the neural network architecture is the biological learning systems, which are inherently dynamic [1]. However, the common practice of building and training a deep neural network (DNN) differs from its biological counterpart. DNN can be defined as a feed-forward artificial neural network with more than one hidden layer. After defining a task, experts search for a DNN architecture that is usually over-parameterized and then initialize the model parameters to start the training process with annotated data.

To bridge the gap between the performance requirement of complex DNN tasks and the resources available on the target computing platforms, various model compression techniques have been proposed. The objective of these algorithms, including binarization, ternarization, quantization, and pruning [2]–[5], is to reduce the computational complexity of a well-trained model before deployment while maintaining similar accuracy. In fact, the success of these methods in itself is a proof that these networks are usually significantly over-parameterized. Most of these techniques are performed after the training process, and they do not take future knowledge into consideration and hence are not directly applicable to online learning or incremental learning scenarios [6]. However, many real-world scenarios, such as adaptive facial recognition systems [7], security oriented tasks such as malware classification [8]–[10], and spam detection [11]–[13] generally require periodically updating their decision models to adapt to varying application contexts.

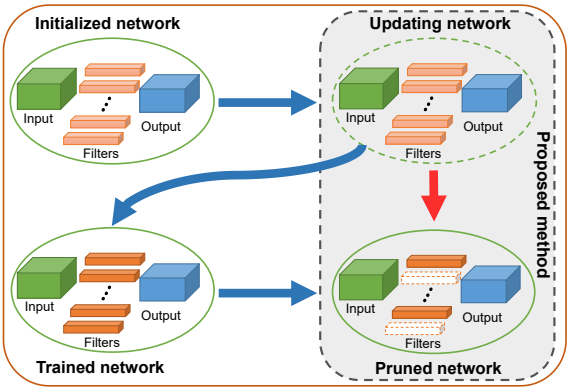

Fig. 1. Conventional pruning methods apply to a well-trained model to obtain a pruned model. In contrast, the proposed method integrates pruning into training.

In this paper, we propose a novel method to accelerate training by removing redundant and less important parameters with genetic algorithms to achieve a similar pattern as biological learning, i.e., **learn faster as learning progresses**. This concept is also supported by recent findings that sub-networks can achieve the same accuracy as initially over-parameterized for training and then pruned networks, with larger training epochs [14], [15]. Unlike the traditional compression algorithms that target a well-trained model with user-defined parameters (e.g., threshold, number of pruning iterations, compression rate) to remove the redundancies [16], our proposed algorithm integrates with the backpropagation and self-regulates the pruning operations. The concept is illustrated in Figure 1. The proposed method can be deployed immediately after initialization or in case of online learning.

To better align with biological learning, the proposed method adopts a genetic-based algorithm, which conducts a

search on the network to provide a reliably pruned network. Besides the "importance" of parameters that have been used in guiding pruning in conventional algorithms, we consider the "stability" of parameters as another critical characteristic in determining parameters to prune. Intuitively, we are looking for stable and unimportant parameters/filters to prune, which we expect to have minimal impact on the model performance. The definitions are given below:

- **Importance** ($\mathcal{I}$) is proportional to the magnitude of the weight of a parameter. Most methods try to avoid removing parameters with significant weights.
- **Stability** ($\mathcal{S}$) of a parameter refers to the changes of its gradient. If the gradient of a parameter becomes smaller over time, it becomes more stable.

Our proposed algorithm is capable of learning the sparsity of the network and modifying the network structures as training progresses. Besides, the genetic-based approach enables wide applicability of training scenarios by also employing training data to decide the pruning criteria. Note that the existing model compression techniques can still be applied to further accelerate the inference after training.

Contributions. The main contributions of this paper are summarized as follows:

- We propose a genetic-based joint dynamic pruning framework to accelerate learning by considering both the importance and stability of parameters.
- The proposed framework allows the model to be trained in the same number of epochs as a conventional training method while achieving minimal loss of accuracy.
- We show that the proposed framework is applicable to both batch learning and online learning scenarios.

II. Methodology

A. Preliminaries

We define the parameters of any convolutional layer l as $\{W^l \in R^{n_O^l \times n_I^l \times k \times k}, 1 \leq l \leq L\}$, where n_I^l, n_O^l, and $k \times k$ represent the number of input channels, number of output channels and kernel size of l-th convolutional layer. In this way $W_j^l \in R^{n_I^l \times k \times k}$ refers to the j-th filter of l-th convolutional layer. As training continues, the network tries to minimize the loss $\mathcal{L}$. The weight gradients g^l are calculated during the backpropagation. Note that g^l has the same shape as W^l. This paper defines the considered structural pruning problem as finding unimportant subset of filters. Previous works such as SFP [17] define the importance ($\mathcal{I}$) as ℓ_p norm of the parameters:

$$\mathcal{I}_j^l = (\sum_{j=1}^{n_O^l} |W_j^l|^p)^{\frac{1}{p}} \tag{1}$$

According to the value of $\mathcal{I}_j^l$ for each filter j, prior methods typically prune all the filters with $\mathcal{I}_j^l$ below a threshold or define a hard constraint and remove a specific number of unimportant filters.

B. Genetic-based Pruning Algorithm

To address the challenges that we discussed in the previous section, we integrate gradients into our algorithm and define the stability $\mathcal{S}$ as the ℓ_1 norm of the gradients, $\mathcal{S}_j^l = (\sum_{j=1}^{n_O^l} |g_j^l|)$.

Filters with low $\mathcal{S}$ scores imply that these filters are stable enough to be pruned. We also use ℓ_1 norm of W_j^l (i.e., $p = 1$ in Equation 1). By considering both $\mathcal{I}_j^l$ and $\mathcal{S}_j^l$ values, we propose a genetic algorithm for updating the mask M, i.e., from M_t at the t-th iteration to M_{t+1}. The mask has the same dimension as the weight W and its entries can only be either 0 or 1, which represent pruned and preserved parameters, respectively. The final pruned model can be obtained by $W \circ M$, where $\circ$ represents the element-wise product. The steps of the genetic algorithm are presented in Algorithm 1. The notations used are summarized in Table I. Algorithm 1 requires weights W and the sum of three consecutive gradients G to form the first generation, mask M which has the similar shape as W and G to keep track of pruned parameters, and two hyperparameters C and T. Here, the main population presents both weights and gradients. In the following sections, we will present the details about the first generation, fitness computation, conventional genetic operations, and the algorithm's overall flow.

Algorithm 1: Genetic-based joint dynamic pruning and learning

Input: Gradients G, Weight W, Mask M_t, Hyperparameters C, T
Output: M_{t+1}

$Z = G \circ W \circ M_t$
$r = n_O/2$
$A = \sum_{i=1}^{r} Z_i, B = \sum_{i=r+1}^{n_O} Z_i$
$P_A[0:r] = Encode(0,1)$
$P_B[0:r] = Encode(0,1)$
$H = A + B$
for $c = 1...C$ | *Until goal achieved* **do**
  $P_A, P_B \leftarrow Crossover(P_A, P_B)$
  if $\mathcal{B}(0.5 \times \frac{c}{C}) == 1$ **then**
    $P_A, P_B \leftarrow Mutation(P_A, P_B)$
  end if
  $F_A \leftarrow Fit(A, P_A)$
  $F_B \leftarrow Fit(B, P_B)$
  if $(F_A + F_B) \leq T \times H$ **then**
    $M_{t+1} \leftarrow Masking(P_A, P_B, M_t)$
    break
  end if
end for

To ensure that pruned filters are stable before pruning, we go one step further to sample the gradient g for several consecutive iterations and consider the sum of these gradient values G_j^l as an indicator of stability. In our algorithm, we sample the gradients for three consecutive times, based on empirical evidence.

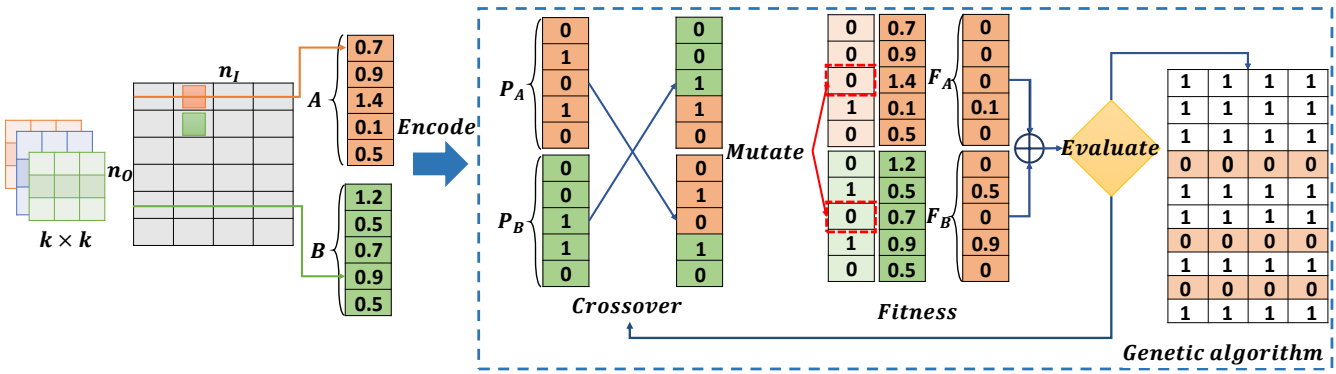

Fig. 2. Flow of the proposed genetic-based pruning algorithm. The first generation reflects the gradients flow and weights of each layer. Crossover and mutation are performed on the encoded P_A and P_B. To evaluate the fitness of the new generation, P_A and P_B are multiplied by A and B, respectively. The sum of these values is compared to the original score. If the evaluation is successful, a mask is produced, which will be used in backpropagation.

TABLE I
NOTATIONS AND HYPERPARAMETERS USED IN OUR ALGORITHM

Notation	Meaning
A and B	The first generations of genetic algorithm
H	Fitness score of the first generation
P_A and P_B	Encoded first generations
C	Maximum number of generations
T	Target

C. First Generation

Figure 2 shows the flow of the he proposed genetic-based pruning algorithm, where A and B represent the first generation of the genetic algorithm. Although we do not explicitly use $\mathcal{I}, \mathcal{S}$, the first generation of the genetic algorithm reflects both of them. If Z is a matrix with the same dimension as W, using $Z_j^l = W_j^l \circ G_j^l \circ M_j^l$ retains the information about the importance and stability at the same time. A small entry in Z represents a small G entry (i.e., stable) and a small W entry (i.e., unimportant), which hence is a candidate for pruning. As shown in Figure 2, after a row-wise addition on Z is performed, we divide the results into two vectors and store them in A and B.

For the $Encode()$ operation, we randomly make two vectors of ones and zeros with the same size as A, and B, which are denoted as population P_A and P_B in the genetic algorithm. The primary operations are performed on P_A and P_B, which are binary vectors.

D. Fitness Computation

Computing the fitness of a population is a vital component of the genetic algorithm. The fitness of the first generation can be easily computed using the first generation A and B. We store this value in H and use it in the evaluation step. H is used to ensure that the pruning candidates' score does not diverge from the first generation score. After that, we compute the fitness of the pruning candidates for each generation, using $Fit()$, as in line 11 of Algorithm 1. The first argument of $Fit()$ is the first generation, and the second argument is the new candidates.

$$F_A = A \circ P_A \tag{2}$$

$$F_B = B \circ P_B \tag{3}$$

$(F_A + F_B)$ is used to evaluate the fitness of the pruning candidates. It is a fast operation that involves pair-wise multiplication and addition.

E. Crossover

Crossover is a popular method in genetic algorithms to explore the solution space. There are different ways of performing crossover [18]. One possible way is to choose two crossover points $q1$ and q_2 in P_A and then swap the chosen subset $P_A[q_1 : q_2]$ with $P_B[q_1 : q_2]$. However, this operation requires several steps for each generation, which might slow down the entire algorithm. Instead, we perform crossover, denoted by function $Crossover()$ as in line 7 of Algorithm 1, by choosing one crossover point of q. As illustrated in Figure 2, a random index in P_A is selected. Then, P_A is swapped with P_B from the beginning to the crossover point.

F. Mutation

Typical mutation operations involve toggling a single bit or shuffling several bits. However, in our work, the mutation operation is designed to expedite the finding of pruning candidates. In this step, $Mutation()$, two points a and b from the population are randomly selected, such that $a < b$. Then members of the candidates are set to zero $P_A[a : b] = 0$, $P_B[a : b] = 0$. The mutation frequency depends on the mutation step. The probability of the mutation step increases as the number of generations continues to grow [19]. For each generation, we draw a binary random number 0 or 1 from a Bernoulli distribution, using $\mathcal{B}(h \times \frac{c}{C})$. As c becomes closer to C, the probability increases. In our experiments, we use $h = 0.5$. If the result of the draw was one, the algorithm performs $Mutation()$. Otherwise, it moves on to calculate the fitness of the evaluation.

G. Evaluation

In this step, the algorithm decides if the current population P_A and P_B can be used for pruning or if it needs to continue searching for a new population. There are two challenges that this step will address. The first challenge is to ensure that by eliminating candidates in P_A and P_B, the training process would not be disrupted. By disruption, we mean a sudden and significant increase in loss. The second challenge is to avoid over-pruning the network in training, which is particularly important for retraining and online learning scenarios.

The first concern can be alleviated by choosing a reasonable constraint. As we can see in Algorithm 1, line 12, the Target T is used as the constraint. A new generation's fitness score noted as (F_A, F_B) should be less than T times the original fitness score H. If we run the genetic algorithm without this constraint, the algorithm will likely prune important filters and significantly reduce the accuracy.

To address the second challenge, we introduce another variable $ratio$, which represents the pruned parameters to the total number of parameters. As the pruning continues, the initial constraint needs to be adjusted, and we use α to adjust it. In the beginning, we use the initial target $T = 0.1$. This means if the $(F_A + F_B)$ is greater than $0.1 \times H$, the candidates' fitness score is too high, and removing them would hurt the accuracy. Otherwise, the pruning candidates are good candidates, and we can continue to $Masking()$. After several pruning rounds, when $ratio$ passes a threshold, the target is multiplied by α. The threshold reduction step is repeated once more in case one of the layers achieves a high pruning ratio. At the same time, as more filters are removed, the value of remained filters becomes too important, and the candidate population will not be pruned unless the candidates are very stable, G_j^l are close to zero, and their weights W_j^l are also close to zero. Although we do not explicitly define a hard constraint for the pruning ratio, theoretically, the algorithm hasn't stopped, but pruning is limited. We tested our algorithm with two different α values, 0.1 and 0.01. At any point, if the evaluation determines suitable candidates are found, the algorithm moves to $Masking()$. In the $Masking()$, we have the pruning candidates P_A, P_B, and the previous mask M_t. Ones in P_A, P_B are the pruning candidates. To update the mask, we toggle the P_A and P_B, concatenate them and extend the result into a matrix with the same size as M. We can perform an element-wise multiplication to get M_{t+1}.

H. Joint Dynamic Pruning and Learning

The main operations of the genetic algorithm are binary inside the for-loop, including crossover and mutation. These operations are integrated into backpropagation, as they would not slow down the training process. Because the algorithm finds suitable candidates, there is no need to keep running it over and over. We run the algorithm every D batches in our experiments, which can also accelerate the entire training.

Algorithm 2 presents the overall flow of our proposed joint dynamic pruning and learning algorithm. After parameter W initialization, we also initialize our mask M. For each epoch, parameters W get updated through stochastic gradient descent. G gets updated using g_t, M_t. Based on the value of D, either genetic-based pruning is performed, or it moves on to finish the backpropagation.

Algorithm 2: Joint dynamic pruning and learning

Input: Training samples X, Training labels Y , Epochs E, Hyperparameter D
Output: Pruned model $W \circ M$

Initialize the model parameters W
Initialize the mask $M_0 \leftarrow all\_ones(M)$
for $e = 1...E$ **do**
  for each batch $\subset X$ ... **do**
    Update W with stochastic gradient descent
    for every D batches ... **do**
      $g_t \leftarrow \nabla_x \mathcal{L}(x, y, W)$
      $G \leftarrow G + (g_t \circ M_t) - (g_{t-3} \circ M_{t-3})$
      $M_{t+1} \leftarrow Genetic(G, W, M_t)$ // Algorithm 1
    end for
  end for
end for

III. Evaluation and Results

A. Experimental Settings

1) Dataset: In this section, we first verify our acceleration method's effectiveness on two benchmark datasets CIFAR [20] and ImageNet [21].

2) Neural Network Architectures: We evaluate our algorithm by using multiple neural network architectures such as ResNet-32, 56, and 110 on CIFAR-10, ResNet-18, 34, and 50 on ImageNet.

We compare our results to two recent representative works that can perform pruning in the training phase from scratch [17], [22]. For a fair comparison, we follow the same steps as in [17], [22] to train these ResNet architectures on CIFAR-10 and ImageNet. Offline pruning methods can be

TABLE II
Performance comparison of ResNet architectures on CIFAR-10

Depth	Baseline Acc.	Method	Pruned Acc. (%)	Acc.↓ (%)	Flop ↓ (%)
32	92.63	SFP [17]	92.08	0.55	41.5
		FPGM [22]	92.31	0.32	41.5
		Ours($\alpha = 0.01$)	93.71	-1.08	32.9
		Ours($\alpha = 0.1$)	91.96	0.67	43.2
56	93.59	SFP [17]	92.26	1.33	52.6
		FPGM [22]	92.89	0.70	52.6
		Ours($\alpha = 0.01$)	92.37	1.22	35.9
		Ours($\alpha = 0.1$)	93.19	0.40	40.1
110	93.68	SFP [17]	93.38	0.30	40.8
		FPGM [22]	93.73	-0.05	52.3
		Ours($\alpha = 0.01$)	93.85	-0.17	47.0
		Ours($\alpha = 0.1$)	93.58	0.10	47.1

TABLE III
PERFORMANCE COMPARISON OF RESNET ARCHITECTURES ON IMAGENET

Depth	Baseline Top-1 (%)	Baseline Top-5 (%)	Method	Pruned Top-1 (%)	Pruned Top-5 (%)	Top-1 ↓ (%)	Top-5 ↓ (%)	Flop ↓ (%)
18	70.28	89.63	SFP [17]	67.10	87.78	3.18	1.85	41.8
			FPGM [22]	67.78	88.01	2.50	1.62	41.8
			Ours ($\alpha = 0.01$)	67.82	88.18	2.46	1.45	20.3
34	73.92	91.62	SFP [17]	71.83	90.33	2.09	1.29	41.1
			FPGM [22]	71.79	90.70	2.13	0.92	41.1
			Ours ($\alpha = 0.01$)	72.25	90.56	1.67	1.06	21.2
50	76.15	92.87	SFP [17]	74.61	92.06	1.54	0.81	41.8
			FPGM [22]	75.03	92.40	1.12	0.47	42.2
			Ours ($\alpha = 0.01$)	75.26	92.37	0.89	0.50	25.8

performed after training on top of the model generated by our proposed method, which results in a better inference speedup.

The proposed algorithm does not enforce a hard threshold or specific hyperparameter for the pruning ratio. The only hyperparameter that can impact the pruning ratio implicitly is α since the evaluation target is adjusted by α.

B. Performance on Batch Learning

We present the performance comparison of pruning on batch learning in Table II and Table III for CIFAR-10 and ImageNet, respectively. We use flops, namely floating-point operations, to evaluate the effectiveness of pruning, which has also been widely used as a metric to evaluate the complexity of DNN models [23], [24].

It can be seen from the experimental results that with the reduction of filters, the accuracy of the models is not impacted. This is expected since our algorithm dynamically adapts to the training process and can maintain the same accuracy without extra steps compared to prior works that target pre-trained models. Compared to [17], [22], our algorithm yields the lowest accuracy drop in general on both CIFAR-10 and ImageNet. Overall, our proposed method achieves comparable performance to the state-of-the-art on batch learning. For example, SFP without hard pruning reduces flops by 41.5% with 0.55% accuracy degradation on ResNet-32, while our method achieves 43.2% flop reduction with 0.67% accuracy drop by dynamically pruning the model during batch learning. On ImageNet, we acknowledge that our method has lower flop reduction. However, our method has a wider range of applicability and achieves superior performance in online learning (Section III-C). Compared to prior methods that are optimized for either pre-trained models or batch learning, our algorithm performs well in both batch learning and online learning setting without any modification. We should also note that $\alpha = 0.1$ is less restrictive, which results in more flop reduction. On the other hand, $\alpha = 0.01$ yields better accuracy than the baseline in some cases.

C. Performance on Online Learning

The main advantage of the proposed method can be observed in online learning. As the new data arrive, the gradients tend to change. Since we consider the gradients flow in our dynamic pruning algorithm, the active filters would not be pruned. The pruning slows down after the initial runs to accommodate the new training data. To perform our experiment, we use the CIFAR-10 dataset; unlike batch learning, new data is provided during the training phase.

TABLE IV
PERFORMANCE COMPARISON OF RESNET ARCHITECTURES ON CIFAR-10 IN ONLINE LEARNING SETTING

Depth	Baseline Acc.(%)	Method	Pruned Acc. (%)	Acc. ↓ (%)	Prune Ratio (%)
32	84.86	ℓ_1 norm [3]	75.85	9.01	20.6
		FPGM [22]	80.21	4.65	31.0
		Ours ($\alpha = 0.01$)	83.50	1.36	33.5
56	83.12	ℓ_1 norm [3]	76.68	6.44	21.8
		FPGM [22]	79.89	3.23	30.6
		Ours ($\alpha = 0.01$)	82.74	0.38	36.5
110	83.18	ℓ_1 norm [3]	74.62	8.56	23.5
		FPGM [22]	80.91	2.27	30.1
		Ours ($\alpha = 0.01$)	80.65	2.53	45.7

In our experiments, we compare our method to FPGM [22] and an ℓ_1 norm-based method [3]. Table IV shows the results and comparison to FPGM [22] on CIFAR-10. SFP [17] performs soft pruning, the pruning ratio during training is very small in our experiments. Thus, we did not include the performance of SFP in the tables, as it would not be a fair comparison. There is a total of ten updates of data during training. The initial learning rate is 0.1 and is multiplied by 0.1 in epochs 150 and 225. For the online learning updates, we follow the method used in [25]. Although they used different training epochs for different methods, we kept the training epochs similar for all the methods. To have a fair comparison with FPGM, we limit the FPGM pruning ratio to 30%, which prevents the model from being over-pruned. In case of no restriction on the pruning ratio, there is a significant decline in the network's accuracy. It can be seen from Table IV that our algorithm simultaneously achieves superior performance in both accuracy preservation and pruning ratio.

We also plot the accuracy trends for these ResNet architectures in Figure 3. When new data is available, the loss

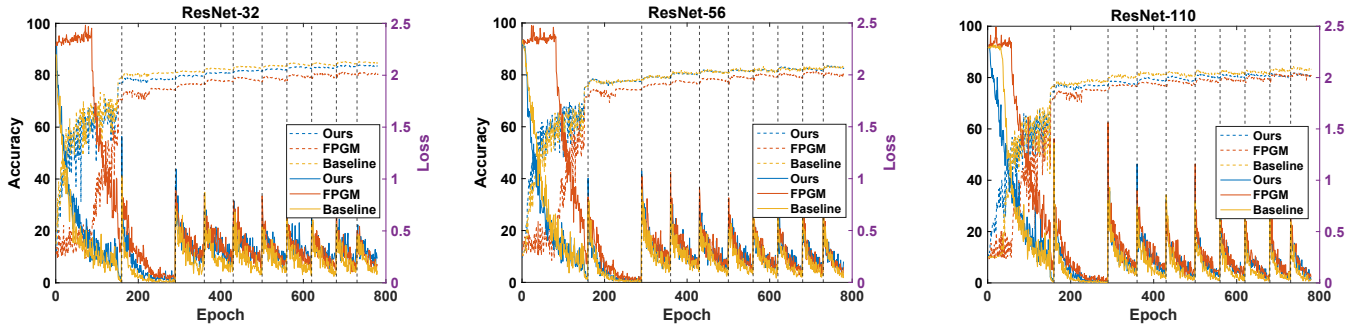

Fig. 3. Comparison of accuracy and loss in online learning setting for different ResNet architectures. The updates of data are performed at epochs 0, 160, 290, 360, 430, 500, 560, 620, 680, 730 (same as in [25]) which are marked by vertical lines. At each update, the loss is increased; however our loss pattern is similar to that of the baseline.

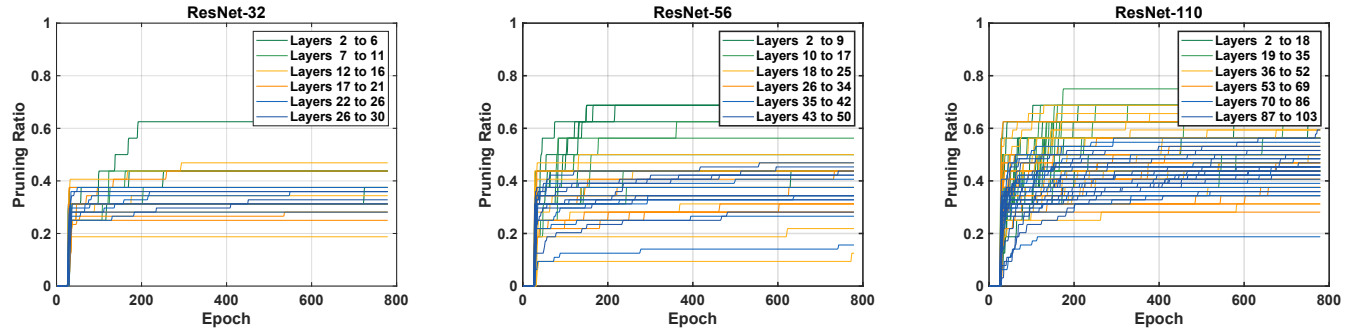

Fig. 4. A comparison of prune ratio in online learning setting for different depths of ResNet architectures. The pruning progress isn't restricted by updates and continues even in late stages of training.

increases, and hence the accuracy drops but will gradually return to similar accuracy as that before the update as training progresses. Compared to FPGM, our proposed method achieves much closer accuracy to the baseline (i.e., without pruning). It is also important to note that even integrated with the proposed dynamic pruning, training converges similar to the baseline. Between epochs 0 to 160 where the network is being trained on the first portion of training data, the baseline and our algorithm behave similarly. The loss starts to go down after several epochs, and this trend continues. Hence, the proposed method can effectively prune the model and accelerate the computation during training without degrading the performance. However, this is not the case with FPGM, where the loss does not go down for a relatively large number of epochs, which might be the reason that the network has a lower accuracy eventually. According to these observations, we argue that the magnitude alone, regardless of the definition of importance (e.g., geometric median, ℓ_1 norm, ℓ_2 norm), is not sufficient for deciding pruning candidates during the training. Thus, integrating stability is essential for performing pruning during training.

To provide more insights into the pruning activities, we plot the pruning ratio for each convolutional layer during training, as shown in Figure 4. We can make some interesting observations, as different layers may have distinctive characteristics in the subsequent stages of online learning. In general, earlier layers tend to achieve a higher pruning ratio than later layers. However, these layers continue to prune gradually as learning further progresses. After a certain number of epochs, pruning activities in earlier layers are significantly reduced (e.g., epoch 200 as shown in the example of Figure 4).

IV. Conclusion

We proposed a joint learning and pruning framework based on genetic algorithms to accelerate DNN. Our algorithm is integrated with the backpropagation to prune filters in convolutional layers without disrupting the learning process. As we noted above, the objective of the proposed method is to apply pruning to both batch learning and online learning. Since pruning on batch learning has been studied in a few prior works already, we calibrate our algorithm more towards online learning. We demonstrated the effectiveness and efficiency of the proposed method in both batch learning and online learning scenarios with a wide range of datasets and neural network architectures. We show the proposed genetic-based algorithm achieves much better performance in comparison to prior works.

Acknowledgements

This work is partially supported by the National Science Foundation award 2047384.

REFERENCES

[1] J. Stiles and T. L. Jernigan, "The basics of brain development," *Neuropsychology review*, vol. 20, no. 4, pp. 327–348, 2010.

[2] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[3] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," in *Proceedings of 5th International Conference on Learning Representations (ICLR)*, 2017.

[4] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 31.1–31.12.

[5] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[6] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, 2018.

[7] B.-F. Wu and C.-H. Lin, "Adaptive feature mapping for customizing deep learning based facial expression recognition model," *IEEE access*, vol. 6, pp. 12 451–12 461, 2018.

[8] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.

[9] J. Clements and Y. Lao, "Backdoor attacks on neural network operations," in *Proceedings of IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2018, pp. 1154–1158.

[10] J. Clements and Y. Lao, "DeepHardMark: Towards watermarking neural network hardware," in *The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[11] G. Jain, M. Sharma, and B. Agarwal, "Spam detection in social media using convolutional and long short term memory neural network," *Annals of Mathematics and Artificial Intelligence*, vol. 85, no. 1, pp. 21–44, 2019.

[12] B. Zhao and Y. Lao, "CLPA: Clean-label poisoning availability attacks using generative adversarial nets," in *The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[13] J. Zhang, X. Chen, M. Song, and T. Li, "Eager pruning: Algorithm and architecture support for fast training of deep neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 292–303.

[14] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proceedings of 7th International Conference on Learning Representations, (ICLR)*, 2019.

[15] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *Proceedings of 7th International Conference on Learning Representations, (ICLR)*, 2019.

[16] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[17] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 2234–2240.

[18] B. Vaissier, J.-P. Pernot, L. Chougrani, and P. Véron, "Genetic-algorithm based framework for lattice support structure optimization in additive manufacturing," *Computer-Aided Design*, vol. 110, pp. 11–23, 2019.

[19] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on evolutionary computation*, vol. 3, no. 2, pp. 124–141, 1999.

[20] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and F.-F. Li, "ImageNet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[22] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.

[23] J.-H. Luo and J. Wu, "Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognition*, vol. 107, p. 107461, 2020.

[24] W. Hua, Y. Zhou, C. De Sa, Z. Zhang, and G. E. Suh, "Boosting the performance of cnn accelerators with dynamic fine-grained channel gating," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, p. 139–150.

[25] X. Dai, H. Yin, and N. K. Jha, "Incremental learning using a grow-and-prune paradigm with efficient neural networks," *IEEE Transactions on Emerging Topics in Computing*, 2020.