An Evolution Approach for Pre-trained Neural Network Pruning without Original Training Dataset

Toan Pham Van, Thanh Nguyen Tung, Linh Bao Doan, and Thanh Ta Minh

***Abstract*—Model pruning is an important technique in real-world machine learning problems, especially in deep learning. This technique has provided some methods for compressing a large model to a smaller model while retaining the most accuracy. However, a majority of these approaches require a full original training set. This might not always be possible in practice if the model is trained in a large-scale dataset or on a dataset whose release poses privacy. Although we cannot access the original training set in some cases, pre-trained models are available more often. This paper aims to solve the model pruning problem without the initial training set by finding the sub-networks in the initial pre-trained model. We propose an approach of using genetic algorithms (GA) to find the sub-networks systematically and automatically. Experimental results show that our algorithm can find good sub-networks efficiently. Theoretically, if we had unlimited time and hardware power, we could find the optimized sub-networks of any pre-trained model and achieve the best results in the future. Our code and pre-trained models are available at: https://github.com/sun-asterisk-research/ga_pruning_research.**

***Index Terms*—Genetic algorithm - GA, model compression, data-free learning.**

I. Introduction

A. Overview

Most state-of-the-art deep neural networks (DNNs) are compute-intensive and require a lot of storage. Due to privacy, latency, and other issues, these computations are gradually moving to the edge. Network pruning is a widely used approach for achieving smaller models with lower computational costs and energy consumption of DNNs, therefore, they can effectively run on edge computing platforms. Pruned models have smaller sizes to fit in edge devices' memory, such as smartphones, security cameras. Structured pruning can even utilize hardware capabilities for reducing latency of models [1].

Many previous works have been proposed for pruning deep neural networks. Based on the number of steps, existing methods can be divided into two categories, i.e., one-shot [2] and iterative [3], [4]. According to the pattern used to prune models, they can be divided into two categories: structured based models [4], [5], [6] and unstructured based models [3].

[1]Manuscript received November 30th, 2021; revised January 7, 2022. This work was supported in part by Sun-Asterisk Inc

Toan Pham Van, Thanh Nguyen Tung, Linh Bao Doan are with Sun-Asterisk Inc., Vietnam (e-mail: pham.van.toan@sun-asterisk.com, nguyen.tung.thanh@sun-asterisk.com, doan.bao.linh@sun-asterisk.com).

Ta Minh Thanh is with Le Quy Don Technical University, Ha Noi, Vietnam. He is now with the Faculty of Information Technology, Le Quy Don Technical University (e-mail: thanhtm@lqdtu.edu.vn).

Most existing pruning methods require original training datasets for retraining [7] or fine-tuning [8]. However, due to privacy (e.g., face datasets) and transmission issues, training datasets might be unavailable. In this paper, we propose a method for effectively pruning DNNs without original training datasets using genetic algorithms. We argue that we always need a validation dataset to evaluate a model before using it. Our method exploits such small datasets for our proposed pruning models. Various experiments on popular datasets (MNIST, CIFAR10) have been conducted to show effectiveness and generalization.

B. Our Contributions

Our main contributions are:

1) We propose a method for pruning models without original training datasets using the genetic algorithm. Therefore, we can efficiently reduce the size of DNNs in case of no training data. That makes our proposed technique can be applied to real applications, especially for edge computing.

2) We introduce a novel strategy for fast and efficient evolving of GA. The strategy requires a much fewer number of generations for a similar average fitness score compared to the original genetic algorithm.

3) We demonstrate that the pruned models achieved by our method are not overfitting on validation datasets. Also, based on such results, we can prove that our algorithm is suitable for large pruned models.

II. Related Works

A. Deep Neural Network

1) Basic concepts

A Deep Neural Network (DNN) is an Artificial Neural Network (ANN) with multiple layers between the input and output layers [9], [10]. There are different types of neural networks, however, they always consist of the same components: neurons, synapses, weights, biases, and functions. Mathematically, a deep neural network is defined as:

$$f_{\theta(x)} = f_{\theta_n} \circ f_{\theta_2} \circ f_{\theta_1}(x)$$

where:

$$\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$$

is its parameters. These parameters are in high dimensional space; and the goal is to find a set of values of θ which maximizes the likelihood of observing some data *x* (a.k.a Maximum Likelihood Estimate). Such can be shown as

doi: 10.18178/ijke.2022.8.1.136

follows:

$$\theta^* = \text{argmax}_\theta \mathcal{L}(\theta; x)$$

where θ^* is the optimal parameters in Maximum Likelihood Estimate settings, and $L(\theta; x)$ is the likelihood function. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. In the case of using DNN, a hypothesis is defined in the form of its network architecture. Subsequently, all that is left to do is to learn the parameters of that network through training.

2) Training a DNN

In order to train a DNN, an objective function must be defined to measure the accuracy of the model; and the training process is optimizing this function to improve the model's quality as a result. The objective function can be defined differently depending on the problems we solve. Some common objective functions while training DNN such as cross-entropy [11] for classification problems and mean square error (MSE) [12] for regression problems, and so on. Optimizing this objective function using gradient descent algorithm [13] is a common practice in deep learning. The optimal parameters (weights) are approximated through iterations; in the case of minimizing a loss function L as the objective, the gradient descent works as follows:

$$\theta_i \longleftarrow \theta_i - \eta \frac{\partial \mathcal{L}(\theta; x)}{\partial \theta_i}$$

where η is the learning rate, and θ_i is the i-th element of the vector parameter θ.

B. Model Pruning

Model compression is one of the most important techniques in deploying neural network models to production level. The purpose of the method is to compress large and complex models into a lighter, simpler one without significant loss in accuracy. This technique is especially meaningful when deployed on low-resource edge devices. One of the typical model compression techniques is model pruning. Enormously trained model in deep learning contains a large amount of redundancy [14] in the form of unimportant weights that have little contribution to the final output. Pruning is a method of model compression, lightening the architecture by cutting off those unimportant connections, trading a minor loss in quality for performance [14]. Both original and pruned model have the same architecture, with the pruned model being sparser (weights with the low magnitude (values) being set to zeros). In essence, pruning is finding the binary mask for each layer in the original network. Each binary mask variable has the same size as the layer (weight) as shown in Fig. 1. The binary mask is used to determine which weights should be used for training the model. These weights can be computed by defining a pruning type strategy.

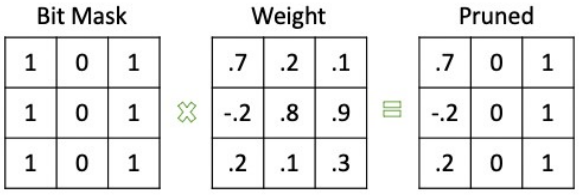

Fig. 1. Binary mask in pruned model.

C. Lottery Ticket Hypothesis

DNNs usually have abundant parameters. However, not all of its parameters are considered useful. Frankle et al. [7] show that there exist sub-networks with a much fewer number of parameters than original networks that can still achieve similar test accuracy. Their method conjectures that for a moderate-sized network f and randomly-initialized parameter $w \in R^d$, there exists a sparse subnetwork f, given by configuration $m \in 0,1^d$, $\|m\|_0 << d$ that can be trained from $w'(0) = m \odot w^0$ to perform comparably to trained versions of the original model f.

However, we cannot start with these subnetworks since they give lower accuracy. Instead, a full original network is needed for effective training. A smaller subnetwork can be uncovered by applying algorithms such as model pruning. Retraining [7] or fine-tuning [8] might be needed to recover accuracy.

D. Genetic Algorithm

Genetic algorithms are one of the important algorithms of evolutionary computing. It is biologically inspired. Darwin's theory of evolution states that all species of organisms arise and develop through natural selection, inherited variations that increase the individual's ability to compete, survive, and reproduce. Through natural selection, the fittest individuals are selected, and they produce offspring. The best individuals will be selected to keep in the next generations. If the parents have higher fitness, their offspring tend to have more chances of survival. Genetic algorithms also learn from this idea. Derived from an initial population including a randomly initialized set of individuals. Through operations such as selection, crossover and mutation, the new population will be created after each generation. Each individual will have a fitness score measuring the quality of it with the current problem.

This algorithm can be used to solve optimization and search problems. Some applications of genetic algorithms can be mentioned such as vehicle routing problems [15], deep learning hyper-parameter optimizations [16], neural network weight optimizations [17] [18], and so on.

III. Proposed Methods

In this section, we present our model pruning method using genetic algorithms. We describe how solution representation, fitness score, initial population initialization, selection, genetic operators, and termination conditions in GA are employed in our approach. We also present our strategies for evolving quickly and efficiently.

Consider a trained neural network $f(x; \theta)$. Let m be a $m \in \{0,1\}^{|\theta|}$ mask for pruning, then the pruned model will be $f(x; m \odot \theta)$. Let p be the desired model sparsity, then our problem becomes finding m such that the pruned model with $p\%$ of the weights retained results in similar accuracy as the trained model.

Solution representation. We use s as a score of m which has the same dimensions as m. The corresponding position of top $p\%$ largest magnitude in scores is 1's positions in m, the remaining are 0. In our approach, s is considered solution representation. This method is illustrated in Fig. 2.

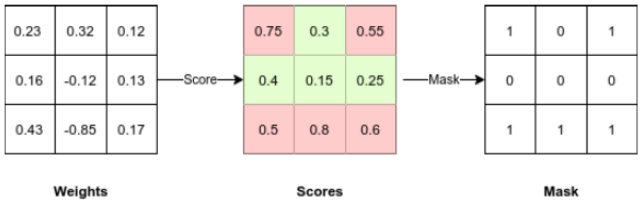

Fig. 2. Solution representation with p=50

Fitness score. In this paper, we demonstrate our method on small datasets. Accuracy is the main metric for these two datasets. So, we use choose this metric as a fitness score to evaluate each individual.

Initial population initialization. The larger population size is, the fewer generations evolve. We set the initial population size (first generation) to 50 to prioritize the number of generations at the early stage. All individuals in the first generation are randomly initialized.

Selection. Parents are selected from individuals in the current generation to produce offspring. Individuals with high fitness scores are more likely to be selected. We use square of fitness scores as a metric for selection to increase discrepancy. This way slightly speeds up evolving compared to using fitness scores.

Crossover. There are three popular variants of crossover. They are single-point, two-point, and uniform crossover. In our method, single-point crossover is adopted. We first flatten then choose crossover points from uniform distribution.

Mutation. Every gene has a small probability to be mutated. The mutated genes are replaced by new values which are uniformly distributed over $[-1,1)$. Evolving process begins with a mutation rate of 0.1 and decreases gradually as fitness scores increase.

Termination conditions. The search space for finding sub-networks is huge. Genetic algorithms often terminate when a predefined condition meets. We end evolving process when the average of fitness scores is no longer increasing after 20 generations

Control mutation rate. The control mutation rate is crucial to achieving good results. A high mutation rate tends to diversify possible solutions at early generations while destroying good solutions at final generations. We control the mutation rate based on the average fitness scores of the latest generation. In general, we decrease mutation rate as fitness scores increase.

Control population size. Large population sizes also diversify possible solutions. However, in earlier generations, with a high mutation rate, large population sizes in these generations might cause wasting our time on inefficient solutions. We change population size based on the average fitness scores of last generations. In the final generations, it is harder to improve already good solutions. Increasing population size alleviates this issue. In general, we increase population size as fitness scores increase.

Retain best solutions. We bring the top 50% best solutions from the previous generation to the next. This brings out several benefits. Firstly, it saves 50% computations each generation. Secondly, it restores accuracy of the original network with much fewer generations (compared to non-retained methods). We also reported subnetworks with higher accuracy than the original network.

The overall workflow of our proposed method is illustrated in Fig. 3.

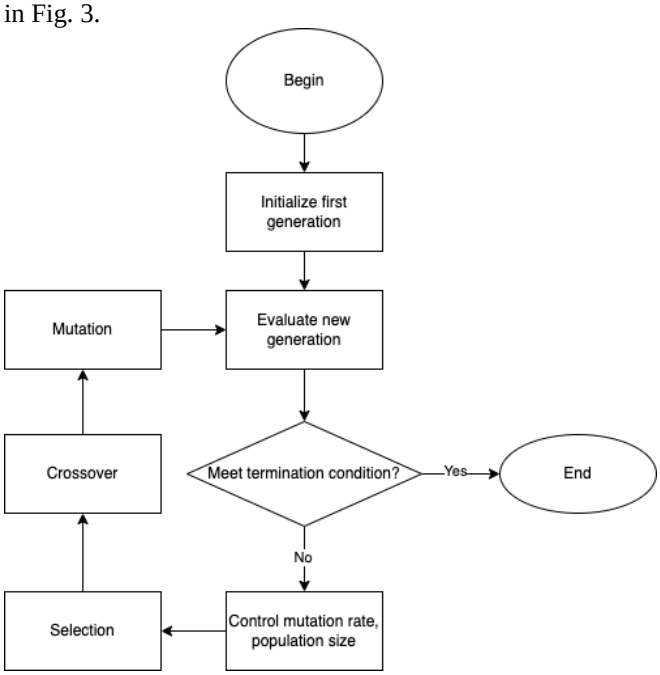

Fig. 3. Overall workflow of our proposed method

IV. Experiments

A. Models Settings

In this section, we report different experimental results of our methodology. Our team conducted multiple research experiments on the base datasets.

With the purpose of assessing our theory, we employ simple convolution neural networks in image classification tasks. Typical vanilla convolutional network architecture consists of three types of layers: convolution, pooling and fully connected layers. Our compact architecture is formulated by multiple layers, each is a stack of two convolution layers followed by max pooling. The output is then followed by three fully connected layers. These networks were trained from scratch and used as our reference networks in our experiments. They were trained with large batch size and optimized using stochastic gradient descent. We experiment pruning models on two types of layers: convolution and fully connected.

B. Dataset

To demonstrate the elaboration of our theory, we experiment multiple approaches on the base dataset MNIST [19] and CIFAR10 [20]. MNIST dataset contains 60,000 images in the training set and 10,000 patterns in the testing set, each of size 28x28 pixels with 256 gray levels. CIFAR-10 dataset has the same amount of training and testing RGB images in 32x32 resolution.

C. Training Scenarios

In the initial experiment, we evaluated our method with a vanilla convolution neural network on MNIST. Same training pipeline also applied for the CIFAR-10 dataset with a more complex model.

MNIST reached 98% on the test set while this number in CIFAR10 is 82%, a good enough number to examine our proposed hypothesis. After training the root model to optimal

point, we applied pruning techniques with strategies that will be explained in the next section.

D. System Configuration

Our experiments are conducted on a computer with Intel Core i7 9700K Turbo 4.9GHz, 32GB of RAM, GPU GeForce GTX 2080 Ti, and 1TB SSD hard disk.

V. Experiment Results

A. Ablation Study

In this section, we show the effects of each component in our methods. Mutation rate control and population size control are crucial in experiments. The details are shown in Table I and II. Comprehensive results of random pruning with different settings are reported in Table III and IV. The metric used for comparisons is average fitness scores at generation 250 for MNIST and 150 for CIFAR10. Convergence comparisons between the base strategy and our proposed Strategy5 are illustrated in Fig. 4 and Fig. 5.

TABLE I: Mutation Rate and Population Size Control Strategies on MNIST

Avg. fitness score	Mutation rate	Population size
Initial	0.15	50
0.8	0.1	100
0.85	0.05	150
0.9	0.025	300
0.95	0.01	600

TABLE II: Mutation Rate and Population Size Control Strategies on CIFAR10

Avg. fitness score	Mutation rate	Population size
Initial	0.2	30
0.8	0.15	60
0.85	0.1	100
0.9	0.05	200
0.95	0.01	300

Increase discrepancy. We use weighted random choice for choosing parents to produce offspring. Two settings are demonstrated for comparison are Strategy1 and Strategy2.

TABLE III: The Result of Ablation Study on MNIST (At Generation 250)

Strategy	Squared fitness	Retain 20%	Retain 50%	Avg. fitness	Best fitness
Strategy1				0.9443	0.9697
Strategy2	✔			0.9569	0.9728
Strategy3		✔		0.9871	**0.9905**
Strategy4			✔	0.9870	0.9881
Strategy5	✔		✔	**0.9883**	0.9900

TABLE IV: The Results of Ablation Study on CIFAR 10

Strategy	Squared fitness	Retain 20%	Retain 50%	Avg. fitness	Best fitness
Strategy1				0.6383	0.6493
Strategy2	✔			0.6404	0.6613
Strategy3		✔		0.7003	0.7074
Strategy4			✔	**0.7114**	0.7184
Strategy5	✔		✔	0.7096	**0.7197**

Strategy1 takes fitness scores of individuals as weights. Strategy2 takes squared fitness scores as weights. Squared fitness scores increase weight discrepancy. This causes the better individuals to have even more chances of selection. Especially, at final generations, when fitness scores of individuals are very close to each other, using squared fitness scores could be a good tiebreaker.

Retain best solutions. In the evolving process, there is no guarantee that the fittest individuals are selected. Additionally, producing an entire new population is computationally expensive when population size is large. For reducing computational cost while maintaining population size, we experiment retaining top 20% and 50% fittest individuals. These two methods both improve fitness scores. As shown in Table III and IV, Strategy3 and Strategy4 achieve similar fitness scores. Strategy4 is also adopted in our method since it saves more computational cost.

B. Checking Generalization

Generalization is the model's ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model. This is a crucial characteristic in model evaluation. We run re-evaluation of our pruned models which evolved on test set on both training set and testing set to check the generalization.

As illustrated in Fig. 6 and Fig. 7, the better pruned network on test set the better accuracy on training set. In earlier generations, the accuracy gaps between training set and test set were small. These gaps increase when approaching the accuracy of the original trained model. At generation 250, accuracy of the best pruned network reaches accuracy of original network. After that, there are slow and small improvements.

C. Results

We successfully pruned 50% weights trained models on MNIST and CIFAR10 without training data. Our pruned models on MNIST do not compromise accuracy score. Table III shows the results of ablation study on MNIST at generation 250. In overall, applied strategies generally help increase the accuracy score. The score of the base pruned method is 0.9443 on average and reaches the peak of 0.9697. The highest number can be found in Stategy5 find, with the best score of 0.9900.

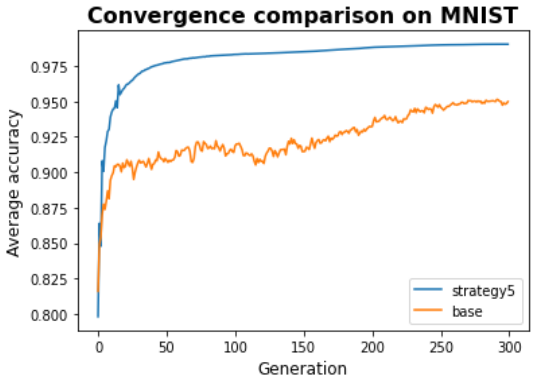

Fig. 4. Convergence comparison between base strategy and Strategy5 on MNIST

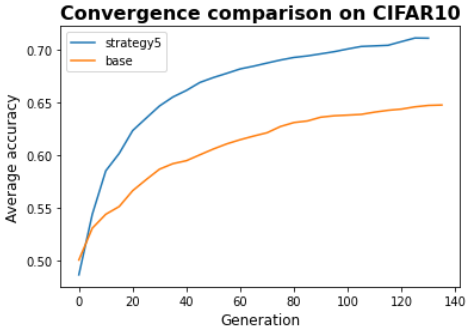

Fig. 5. Convergence comparison between base strategy and Strategy5 on CIFAR10

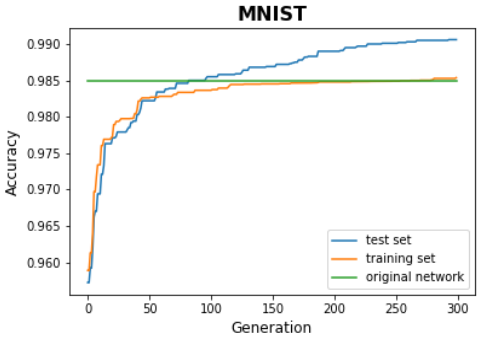

Fig. 6. Evaluate best pruned networks through generations on MNIST

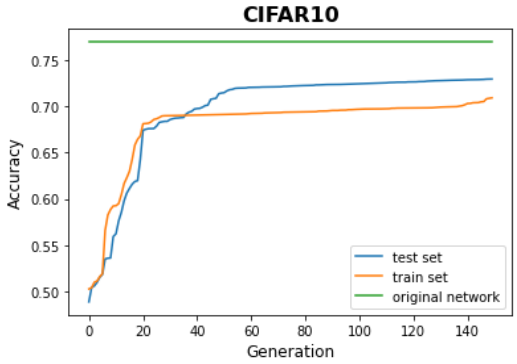

Fig. 7. Evaluate best pruned networks through generations on CIFAR10

VI. Conclusion and Future Work

In this work, we conducted various extensive experiments to show the effectiveness of genetic algorithms in achieving good performance of pruned networks. We conclude that compared to traditional fine-tuning, genetic algorithms are another promising approach to recover performance decrease due to previous pruning methods.

The hypothesis for this experiment is based on intuition of train-data free pruning purposes. We are looking forward to extending our work in the future in terms of scale optimization and larger dataset.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Toan Pham Van came up with the ideas and designed the algorithm; Thanh Nguyen Tung, Linh Bao Doan conducted the research and reported experimental result; Thanh Ta Minh gave technical advice and reviewed the paper; all authors had approved the final version.

Acknowledgment

This work is partially supported by Sun-Asterisk Inc. We would like to thank our colleagues at Sun-Asterisk Inc for their advice and expertise. Without their support, this experiment would not have been accomplished.

References

[1] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," arXiv preprint arXiv:2104.08378, 2021

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015

[3] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning Convolutional neural networks for resource efficient inference," arXiv preprint arXiv:1611.06440, 2016

[4] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[5] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017

[6] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in Neural Information Processing Systems*, vol. 29, pp. 2074–2082, 2016.

[7] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," arXiv preprint arXiv:1803.03635, 2018.

[8] N. Liu, G. Yuan, Z. Che, X. Shen, X. Ma, Q. Jin, J. Ren, J. Tang, S. Liu, and Y. Wang, "Lottery ticket preserves weight correlation: Is it desirable or not?" in *Proc. International Conference on Machine Learning*. PMLR, 2021, pp. 7011–7020.

[9] Y. Bengio, Learning Deep Architectures for AI. Now Publishers Inc, 2009.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[11] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Proc. 32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[12] D. M. Allen, "Mean square error of prediction as a criterion for selecting variables," *Technometrics*, vol. 13, no. 3, pp. 469–475, 1971.

[13] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arXiv:1609.04747, 2016.

[14] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017.

[15] B. M. Baker and M. Ayechew, "A genetic algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 30, no. 5, pp. 787–800, 2003.

[16] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proc. the Workshop on Machine Learning in High-performance Computing Environments*, 2015, pp. 1–5.

[17] S. Ding, C. Su, and J. Yu, "An optimizing bp neural network algorithm based on genetic algorithm," *Artificial Intelligence Review*, vol. 36, no. 2, pp. 153–162, 2011.

[18] J. N. Gupta and R. S. Sexton, "Comparing backpropagation with genetic algorithm for neural network training," *Omega*, vol. 27, no. 6, pp.679–684, 1999.

[19] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[20] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (Canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

Pham Van Toan received the Bachelor of Engineering degree and the Master's degree of Computer Science from Hanoi University of Science and Technology, in 2016 and 2021, respectively. He is Head of AI Research in Sun Asterisk Inc. His research interests lie in the area of deep learning, model compression techn\mque, and computer vision.

Thanh Nguyen Tung received his B.S from Hanoi University of Science and Technology in electronics and telecommunication engineering.
He is currently a Machine Learning Research Engineer in R&D Lab, Sun Asterisk.

Linh B Doan received his B.S from National University of Civil Engineering in Information Technology. His research interests are machine learning, computer vision.
He is currently a Machine Learning Research Engineer in R&D Lab, Sun

Ta Minh Thanh received the Bachelor of Engineering degree and the Master degree of Computer Science from National Defence Academy, Japan, in 2005 and 2008, respectively. He was lecturer of Le Quy Don university, Ha Noi, Viet Nam from 2005. In 2015, he received the Ph.D degree in Computer Science from Tokyo Institute of Technology, Tokyo, Japan. He received the standards of associate professor title in 2019.
He is also the member of IPSJ Japan and IEEE. His research interests lie in the area of watermarking, network security and computer vision.