Lightweight CNN Frameworks and their Optimization using Evolutionary Algorithms

Yangyang Chang
Dept. of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN USA
chan1729@umn.edu

Gerald E. Sobelman
Dept. of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN USA
sobelman@umn.edu

2022 International Electrical Engineering Congress (iEECON) | 978-1-6654-0206-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/iEECON53204.2022.9741692

***Abstract*— This paper presents novel frameworks for efficient and lightweight convolutional neural networks that are suitable for use in embedded system applications. Population-based metaheuristic approaches including the genetic algorithm, cuckoo search, the multifactorial evolutionary algorithm and a proposed hybrid approach are used to optimize their performance on image classification tasks. The methods utilize small population sizes without requiring weight-sharing or a surrogate function, and both binary and integer encoding methods are applied in the optimization. The results from these various strategies are evaluated using metrics of computation time and classification accuracy. The multifactorial-based approach is found to give the highest classification accuracy and it requires only a moderate evaluation time. Also, comparisons with prior approaches demonstrate that these methods show a favorable tradeoff between accuracy and computational cost.**

Keywords*— *network architecture search, multifactorial evolutionary algorithm, genetic algorithm, cuckoo search, convolutional neural networks

I. INTRODUCTION

Recently, Network Architecture Search (NAS) [1] has become an important methodology because manually designing complex neural network architectures requires significant effort. In prior research using evolutionary algorithms (EAs) [2, 12-14], each potential neural network structure is encoded as an individual in a population. The evolutionary process utilizes the exchange of information between individuals or random changes within an individual. The optimization procedure favors selecting individuals having the best performance. In image-based applications such as the ones considered here, an individual's fitness value is given by the classification accuracy of the corresponding trained neural network. The diversity of information during the search process increases with the size of the population, but the computation time becomes significant when a large population size is used. For example, [13] trains 100 individuals in parallel on 100 GPUs in each generation.

The frameworks proposed in this work use the residual-inception concept [6-8], in which multiple input paths are concatenated to form an output tensor. Two novel convolutional neural network (CNN) frameworks are considered, a two-stage network and a three-stage network. Several EAs and encoding methods are applied to these frameworks and the trade-offs between network depth and computational load are evaluated. A structure is encoded based on its interconnections, the number of layers and output channels, and the size of the convolutional filters. Compared with [2], which only considers the genetic algorithm with a binary encoding, we evaluate a wider range of algorithm and encoding combinations. In particular, the genetic algorithm [5], cuckoo search [3, 4, 9], a novel combination using Lévy-flight from cuckoo search and crossover from the genetic algorithm that we call Lévy- flight-crossover, and a multifactorial evolutionary algorithm [16, 17] are applied and evaluated.

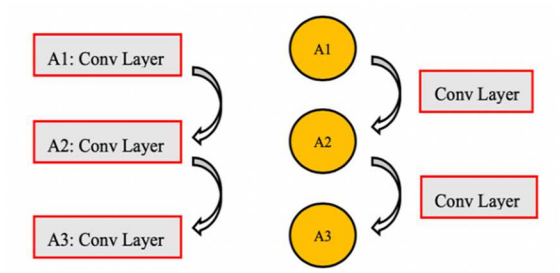

Fig. 1. (Left) Structure design of [2]. (Right) Structure design of this paper. Here, A1, A2 and A3 represent three nodes, with A1 connected to A2 and A2 connected to A3. Conv Layer is where the convolutional operation takes place. Note that the structure used in [2] performs the convolution at the node, while the structure used in this paper performs the convolution on the connection between nodes

II. CNN FRAMEWORKS

The optimization process considers the interconnections between nodes, the size of the filters and the number of output channels for each convolutional layer. Although other aspects such as the choice of activation function or the dropout rate could also be important, including too many of these would pose a greater burden on an EA's effectiveness, especially when the size of the population is small, as it is in this work.

Note that in [2], the convolution operation is performed within a node. In our approach, as illustrated in Fig. 1, a concatenation operation is performed at a node while the convolution operation takes place between nodes. More specifically, our node receives inputs from convolutional outputs on the connections, whereas the nodes of [2] receive the convolutional outputs from the other nodes.

The details of the two-stage network, as shown in Fig. 2, are as follows. The first stage has five nodes and ten possible interconnections, and there is a possible interconnection between the first and second stages. The second stage has six nodes and fifteen possible interconnections. To avoid overfitting, we apply various dropout rates that have been found to give good results. In particular, the first stage has a 30% dropout rate, the second stage has a 40% dropout rate and the final dense layer has a 50% dropout rate. After the final concatenation node, the max-pooling operation for two-dimensional spatial data is used and a fully-connected layer containing 512 neurons and a leaky rectifier unit (with a parameter value of 0.2) is included. One possible configuration that may be generated during the iterations is also shown in Fig. 2. Note that it has a dangling connection between the first node and the second node, since the output of the second node is not propagated further through the network.

978-1-6654-0206-4/22/$31.00 ©2022 IEEE

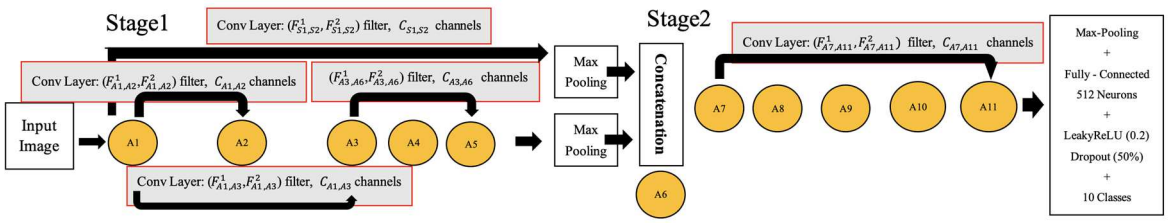

Fig. 2. The two-stage network structure used in this work. The notation "Conv Layer: $(F^1_{x,y}, F^2_{x,y})$, $C_{x,y}$ channels" indicates that there is a convolutional layer between nodes x and y, where the convolutional calculation uses a filter size of $F^1_{x,y}$ by $F^2_{x,y}$ and there are $C_{x,y}$ channels.

The 3-stage network structure is similar to that of the 2-stage network but with an additional stage.

III. Encoding Methods

In binary encoding, we only consider optimizing the connections between the concatenation nodes. A candidate network is encoded as a binary string. Each '1' or '0' bit represents the connection or disconnection between two concatenation nodes, respectively. A specific encoding example is also shown in Fig. 2.

There are two integer encoding methods. In integer encoding #1, the size and number of the filters, as well as the interconnections of the concatenation nodes are specified using integers. In other words, all instances of $F^1_{x,y}$ $F^2_{x,y}$ and $C_{x,y}$ in Fig. 2 are factors to be optimized and are represented as integers. The set of all interconnections in a stage is also specified using a single integer.

After performing simulations using integer encoding #1, we have found that the filter size is of lesser importance in the optimization process. Therefore, in integer encoding #2, the size of the filter is fixed at 3-by-3. Similar to integer encoding #1, an individual is encoded as a fixed-length integer string specifying the connections between concatenation nodes as well as the number of channels in each convolutional layer.

IV. Optimization Algorithms

A. Genetic Algorithm (GA)

GAs have been employed to generate solutions to optimization and search problems by utilizing the operations of selection, crossover and mutation [5]. An initial population of individuals (i.e., candidate solutions) to an optimization problem evolves to create improved solutions. The best individuals are obtained by the selection operator, and new individuals are created through crossover and mutation.

There are several types of selection methods, such as roulette selection, the fitness ratio method, reward-based selection and tournament selection [24]. Here, we use roulette selection for which the selection probability of an individual is determined by its fitness value, which is taken to be the classification accuracy that is obtained.

The crossover operation enhances the ability to do global searching. For each selected individual (i.e., parent), we randomly select another individual with which to perform the crossover so that the newly created individual will include attributes of both of the parents. Unlike the traditional K-point method [1], our method generates only one child and each gene has the same probability for crossover. For a length-L individual, L random binary numbers are drawn uniformly, which determine the crossover point values for each of L genes. If the crossover point value is 0, the child takes its corresponding gene from Parent 1. Otherwise, it takes the gene from Parent 2. Note that we only have to evaluate one new child for each pair of parents per generation. Since fitness evaluation consumes the majority of the computation time in the NAS problem, this approach saves a considerable amount of time compared to the standard approach in which two children are generated. We use a crossover probability of 0.7.

The mutation operation also allows for an individual to break out of a local search region. We use an external L-point mutation method, where L is the number of genes. Each gene takes its value from a specified set of values. Similar to the crossover method, each gene has a random mutation point value and a specified mutation threshold. When the random mutation point value is smaller than the specified mutation threshold, this gene is changed to a value selected randomly from its gene set. A mutation probability of 0.05 is used here.

B. Cuckoo Search (CS)

CS was developed based on the behavior exhibited by that bird species [9]. The eggs of cuckoo birds are laid in nests constructed by birds of another species (called the host), particularly those species having eggs of similar appearance [25]. The cuckoo eggs hatch first, and the cuckoo chicks then push the host eggs out of the nest, thereby effectively taking over the nest.

Assuming there are many host nests and each nest only has one host egg, the basic CS algorithm consists of the following four rules [9]: 1) A cuckoo randomly finds a host nest and places one egg in the nest. 2) The cuckoo's egg is discovered with probability $p_{a,cs}$ by a host bird, and then it can either abandon this nest, find the egg of a cuckoo, or build a new nest. 3) Each nest will be evaluated, and the best nest and its egg are carried over to the next generation. 4) All nests are updated using the difference between the best nest and the other current nests for the next iteration. The goal of these rules is to use the cuckoos to replace one bad nest and to modify of all of nests towards having the characteristics of the best nest.

CS is further enhanced using a local change and a global random process known as a Lévy flight [9]. The local random process updates several nests which are found with probability $p_{a,cs}$ and adjusts them according to the differences with the other nests. The Lévy flight provides an efficient method for sampling the global solution space [26] through the use of a random increment size and direction. The balance between local and global searching is controlled by two scaling factors. The local scaling factor, a_1, affects the step size of the local random process, while the second scaling factor, a_2, determines the increment amount of the Lévy flight. The details of the iteration update equations are given in [25].

In this work, we allow nests to contain several eggs. To

TABLE I. ALGORITHM COMPARISON APPLIED TO THE 2-STAGE NETWORK FOR MNIST WHEN THE POPULATION SIZE IS 20

Algorithm	Classification Accuracy (Binary Encoding)
GA	98.83%
CS	99.28%

TABLE II. ALGORITHM COMPARISON APPLIED TO THE 2-STAGE NETWORK FOR CIFAR-10 WHEN THE POPULATION SIZE IS 20

Algorithm	Normalized Evaluation Time Per Generation	Classification Accuracy	
		Binary Encoding	Integer Encoding #1
GA	100%	81.61%	73%
CS	125%	83.44%	76.83%
Lévy Flight + Crossover	170%	82.64%	76.808%

TABLE III. ALGORITHM COMPARISON APPLIED TO THE 3-STAGE NETWORK FOR CIFAR-10 WHEN THE POPULATION SIZE IS 20

Algorithm	Evaluation Time Per Generation	Classification Accuracy (Integer Encoding #2)	Average CPU Wall Clock Time Per Generation
CS	125%	85.67%	0.39 days
Lévy Flight + Crossover	170%	84.66%	0.67 days
MFEA	150%	87.59%	0.53 days

TABLE IV. COMPARISON OF ACCURACY AND NUMBER OF PARAMETERS AGAINST RECENT PRIOR WORK.

Optimized Structure (Ranked by Accuracy)	Classification Accuracy (CIFAR-10)	Number of Parameters (Millions)
ViN [18]	75.26%	0.62
Hybrid PiN [18]	74%	0.99
CCN [18]	83.36%	0.91
Proposed 3-stage network & MFEA w/data augment.	91.55%	0.88
VGG-19 with GradInit [19]	94.71%	20.03
CCT-6/3× 1 [20]	95.29%	3.17
ResNet-50-SAM [21]	97.4%	25
WRN-28-10 [22]	97.73%	36.5
EfficientNetV2-M [23]	99.0%	55

save evaluation time, in the local random process we only evaluate a nest when it is found by the cuckoo. In contrast to the procedure in [25], only 25% of the population per generation is evaluated in the local random process when $p_{a,cs}$ is 0.25. The scaling factor a_1 is set to 1, 0.03 or 0.15 for binary encoding, integer encoding #1 or integer encoding #2, respectively. a_2 is set to 0.15 for binary encoding, 0.2 for integer encoding #1, and 0.15 for integer encoding #2.

C. Lévy-Flight-Crossover

By combining aspects of the GA and CS algorithms, a new algorithm we call Lévy-flight-crossover is proposed. The Lévy-flight is applied first, followed by crossover with a 70% crossover rate. A nest is updated when the new configuration has a better fitness value than its previous one. Roulette selection is used to increase the representation of information from elite individuals. The fitness values of all individuals are stored after each operation to eliminate having to re-evaluate individuals that have already been seen. Lévy-flight requires evaluating all nests and crossover requires evaluation of 70% of the population. Therefore, the number of evaluations in each generation of Lévy-flight-crossover is 170% of the population size. Compared with CS, Lévy-flight-crossover requires 45% more evaluations per generation. On the other hand, it has only one scaling factor to be determined, i.e. the one for Lévy-flight, as opposed to the two scaling factors in CS. Here, we set the Lévy exponent to 1.5.

D. Multifactorial Evolutionary Algorithm (MFEA)

MFEA has been shown to be a powerful method to handle multifactorial optimization problems [17, 27]. MFEA accounts for the coexistence of multiple tasks, where each one is an optimization having its own characteristics [17]. The goal is to take advantage of potential knowledge transfer between the tasks in a way that speeds up the convergence [17, 27]. Thus, how to select suitable tasks becomes a key design issue in an MFEA application.

Recent studies have considered several different forms of MFEA, such as multifactorial particle swarm optimization, multifactorial genetic programming and multifactorial differential evolution [15]. These methods have been applied to both continuous and discrete optimization problems, with good results. [15, 17, 27].

In our application, we utilize two tasks to optimize a CNN structure. We have also included two modifications to the standard MFEA approach. First, for individuals which are good at both tasks, an additional skill factor value is assigned. Second, starting from a population of individuals p_i, we construct a set of task individuals $p'_{i,j}$ where the index j specifies the task number. Each task individual is composed of two parts. One part comes from the individual p_i, while the other part is from the best individual on task j obtained so far. We have found that these modifications lead to performance improvements in both the final result obtained and for the computation time that is required.

V. RESULTS

An Intel® 6-core i9-8950HK processor running at 2.9 GHz with 16GB of RAM, and a single GPU core NVIDIA GeForce® GTX1070 are used in our experiments. The batch size is set at 50, with 70 training epochs used for the CIFAR-10 dataset and 20 training epochs applied in the MNIST dataset.

A. Binary Encoding for MNIST

Table I shows simulation results for binary encoding with the MNIST dataset. Starting with the same population of 20 initialized individuals, CS achieved slightly better results than GA, although both procedures gave very high accuracy.

B. Binary Encoding and Integer Encoding #1 for CIFAR-10

In Table II, simulation results for binary encoding and integer encoding #1 for CIFAR-10 when the population size is 20 are shown. As was the case for MNIST, CS can achieve better results than GA for CIFAR-10. The crossover operator of GA requires a larger population size to achieve high performance, so having only one child in our GA implementation reduces its performance. However, if two children were evaluated, the evaluation times would be far larger than for CS. Compared to our GA implementation, CS, requires 25% more computation time per generation than GA. In addition, the two scaling factors a_1 and a_2 need to be determined prior the simulations.

As shown in the table, the Lévy-flight-crossover algorithm requires 45% more evaluation time per generation, and it achieves a performance similar to CS. However, an advantage of the Lévy-flight-crossover is that only one scaling factor (i.e., for Lévy flight) needs to be determined.

C. Binary Encoding #2 for CIFAR-10

Simulation results comparing three algorithms for the 3-stage network when the population size is 20 are shown in Table III. (GA was not considered in these simulations since the results in the prior section showed that CS and Lévy-flight-crossover gave better results.) The MFEA has the highest classification accuracy among the three algorithms. In the table, we also give the average wall clock time per generation since structures having different initial populations require different simulation times. The MFEA with the 3-stage network was also applied to the MNIST dataset, resulting in a classification accuracy of 99.65%.

Comparing Tables II and III, we can see the benefits of using the 3-stage network with integer encoding #2. The classification accuracy of CS is increased from 83.44% to 85.67%, while the Lévy-flight-crossover improves from 82.64% to 84.66%. The evaluation times and the CPU wall clock time are still quite modest with the 3-stage network.

To further improve the accuracy, we have also tested a data argumentation method [10, 11]. The final classification accuracy obtained (after applying data augmentation) is 91.55% and it uses only 0.88 million parameters.

Finally, in Table IV, we compare the proposed 3-stage network (using the MFEA with data augmentation) against other recently published results. While our accuracy is lower than that of EfficeintNetV2-M, the number of parameters required is much smaller, by a factor of 62.5. On the other hand, our accuracy is significantly higher than that of Hybrid PiN and CCN, even though those two designs require a larger number of parameters. Thus, the proposed network provides a beneficial trade-off of accuracy vs. complexity compared to recent prior work.

VI. Conclusion

In this paper, lightweight convolutional neural network frameworks that are suitable for embedded system applications have been proposed and optimized using evolutionary algorithms. We have found that an MFEA-based approach yields the highest classification accuracy results while requiring only a moderate evaluation time. Another benefit of that algorithm is that there are no predefined hyperparameters to determine. On the other hand, CS executes faster. In our simulations, CS takes only 0.39 days per generation using our laptop GPU. However, two scaling factors must be determined before the simulations can take place. The proposed Lévy-flight-crossover approach only requires one hyperparameter to be determined and gives similar performance to CS, but it requires the most evaluation time among these four algorithms. GA gave the poorest performance, but this is primarily because our population size of 20 is quite small compared to the population sizes that are normally used in practice.

The proposed lightweight 2-stage and 3-stage CNN frameworks are well suited for embedded system applications having limited computational capabilities. For the 3-stage network, the MFEA with the data argumentation technique can achieve 91.55% accuracy with only 0.88 million parameters for CIFAR-10.

References

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search," in Automated Machine Learning, The Springer Series on Challenges in Machine Learning, F. Hutter, L. Kotthoff and J. Vanschoren, Eds., Springer, 2019.

[2] L. Xie and A. Yuile, "Genetic CNN," arXiv:1703.01513, 2017.

[3] X.-S. Yang, Nature-Inspired Metaheuristic Algorithms, 2nd ed., Luniver Press, 2010.

[4] V. Oliveira, R. de Oliveira and C. Affonso, "Cuckoo search approach enhanced with genetic replacement of abandoned nests applied to optimal allocation of distributed generation units," IET Generation, Transmission & Distribution, vol. 12, no. 13, pp. 3353–3362, 2017.

[5] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA. 1998.

[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision," arXiv: 1512.00567, 2015.

[7] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He "Aggregated residual transformations for deep neural networks," arXiv:1611.05431, 2017.

[8] C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," arXiv:1602.07261, 2016.

[9] X.-S. Yang XS and S. Deb, "Cuckoo search via Lévy flights," World Congress on Nature & Biologically Inspired Computing (NaBIC), pp. 210–214, 2009.

[10] E. D. Cubuk, B. Zoph, D.Mane, V. Vasudevan and Q. V. Le, "AutoAugment: Learning augmentation policies from data," arXiv:1805.09501v3, 2019.

[11] E. D. Cubuk, B. Zoph, J. Shlens and Q. V. Le "RandAugment: Practical automated data augmentation with a reduced search space," arXiv:1909.13719v2, 2019.

[12] Y. Chang, G. E. Sobelman and X. Zhou, "Genetic architecture search for binarized neural networks," IEEE 13th International Conference on ASIC (ASICON), 2019.

[13] R. Miikkulainen, et al, "Evolving deep neural networks," arXiv:1703.00548v2, 2017.

[14] Y. Sun, B. Xue, M. Zhang, G. G. Yen and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," IEEE Transactions on Cybernetics, vol. 50, no. 9, pp. 3840–3854, 2020.

[15] L. Feng et al., "An empirical study of multifactorial PSO and multifactorial DE," IEEE Congress on Evolutionary Computation (CEC), pp. 921–928, 2017.

[16] A. Gupta, Y. Ong and L. Feng. "Multifactorial evolution: Toward evolutionary multitasking," IEEE Transactions on Evolutionary Computation, vol. 20, no. 3, pp. 343-357, 2016.

[17] Y. Chang and G. E. Sobelman, "Simpler and enhanced multifactorial evolutionary algorithms for continuous optimization tasks," IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS), 2021.

[18] P. Jeevan and A. Sethi, "Vision Xformers: Efficient attention for image classification," arXiv:2107.02239, 2021.

[19] C. Zhu, et al., "GradInit: Learning to initialize neural networks for stable and efficient training," arXiv:2102.08098v1, 2021.

[20] A. Hassani, et al. "Escaping the big data paradigm with compact transformers," arXiv:2104.05704v3, 2021.

[21] R. Wightman, H. Touvron and H Jegou, "ResNet strikes back: An improved training procedure in timm," arXiv:2110.00476v1, 2021.

[22] A. Rame, R. Sun and M. Cord, "MixMo: Mixing multiple inputs for multiple outputs via deep subnetworks," arXiv:2103.06132v3, 2021.

[23] M. Tan and Q. Le, "EfficientNetV2: Smaller models and faster training," arXiv:2104.00298v3, 2021.

[24] A. Lipowski, "Roulette-wheel selection via stochastic acceptance," arXiv:1109.3627, 2011.

[25] X.-S. Yang, Nature-Inspired Optimization Algorithms, 2nd ed., Academic Press, 2020.

[26] A. M. Reynolds and M. A. Frye, "Free-flight odor tracking in Drosophila is consistent with an optimal intermittent scale-free search," PLoS ONE, vol. 2, no. 4, pp. e354–363, 2007.

[27] K. K. Bali, Y. Ong, A. Gupta and P. S. Tan, "Multifactorial evolutionary algorithm with online transfer parameter estimation: MFEA-II," IEEE Transactions on Evolutionary Computation, vol. 24, no. 1, pp. 69-83, 2020.