Neural Architecture Search for Resource-Constrained Internet of Things Devices

Isadora Cardoso-Pereira
Department of Computer Science
Federal University of Minas Gerais
Minas Gerais, Brazil
isadoracardoso@dcc.ufmg.br

Gisele Lobo-Pappa
Department of Computer Science
Federal University of Minas Gerais
Minas Gerais, Brazil
glpappa@dcc.ufmg.br

Heitor S. Ramos
Department of Computer Science
Federal University of Minas Gerais
Minas Gerais, Brazil
ramosh@dcc.ufmg.br

2021 IEEE Symposium on Computers and Communications (ISCC) | 978-1-6654-2744-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISCC53001.2021.9631535

***Abstract*—The traditional process of extracting knowledge from the Internet of Things (IoT) happens through Cloud Computing by offloading the data generated in the IoT device to processing in the cloud. However, this regime significantly increases data transmission and monetary costs and may have privacy issues. Therefore, it is paramount to find solutions that achieve good results and can be processed as close as possible to an IoT object. In this scenario, we developed a Neural Architecture Search (NAS) solution to generate models small enough to be deployed to IoT devices without significantly losing inference performance. We based our approach on Evolutionary Algorithms, such as Grammatical Evolution and NSGA-II. Using model size and accuracy as fitness, our proposal generated a Convolutional Neural Network model with less than $2\,\mathrm{MB}$, achieving an accuracy of about $81\,\%$ in the CIFAR-10 and $99\,\%$ in MNIST, with only 150 thousand parameters approximately.**

***Index Terms*—Neural Architecture Search, Internet of Things, Resource-constrained devices**

I. Introduction

The traditional method of extracting knowledge from the Internet of Things (IoT) devices happens by sending the data generated in the IoT object for processing in the cloud. Cisco estimates that by 2021, people, machines, and things will generate approximately 850 zettabytes (ZB) of data. It is easy to observe that moving this massive amount of data over the network can yield high cumbersome transmission and monetary costs, in addition to privacy leakage issues [1].

As IoT adoption is increasing rapidly and becoming part of our daily lives, more sophisticated solutions that can produce real-time analytic insights are expected, such as personal assistants and personalized healthcare. Hence, there is a need to process data in the edge of the networks, including in the IoT device itself [2]. This approach presents several advantages, such as low latency, more accessibility, and high-quality real-time decision-making, increasing the operational efficiency of urban environments, which are fast-changing by nature. Due to their state-of-the-art results in several areas, such as computer vision and speech recognition, researchers consider Deep Learning (DL) algorithms for this processing. However, their demand for great computational power contrast with the low processing capacity and energy that IoT devices have, making this alliance unfeasible [1].

In this scenario, we are searching for DL models that are accurate and small enough to be deployed as close as possible to an IoT object. Typically, the development of such models is complex, demanding intense human effort in an iterative trial and error process to find reasonable components to assemble a neural network, such as topology, hyper-parameters, learning algorithms (and its hyper-parameters), among others. The choices of each aspect are interdependent and require a high degree of expertise to avoid poor performance [3]. It led to the emergence of Neural Architecture Search (NAS), that is, semi-automatic systems that seek the best neural network model for solving a problem, with minimal human intervention [4].

This paper proposes a NAS solution that takes advantage of Evolutionary Algorithms (EA) to find neural network architectures with competitive results that can be used in IoT devices by having a small number of parameters (and consequently, a small size).

The main contributions of this work are threefold:

- A novel EA-based algorithm to evolve neural networks from scratch, in order to find DL models that are accurate and small enough to be deployed to resource-constrained devices, as commonly seen in IoT;
- Differently from previous work in NAS for IoT, we use Grammatical Evolution (GE), making our resulting networks more readable. Hence, people not familiar with EA algorithms can also use our approach;
- While previous works use a block-based search on well-known DL models, we start the network from scratch, incorporating less human bias.

The remainder of this work is organized as follows: Section II presents the related work; Section III describes the methodology used; Section IV shows the main results; and Section VI concludes this work.

II. Related Work

Several DL models can achieve similar results, but to implement them as near as possible to the edge of the network, we need to find models with few parameters (i.e., small models). We can find a myriad of hand-crafted solutions to this problem in the literature. For example, in [5], the authors proposed SqueezeNet, a method that uses modified convolutional modules, achieving results comparable to AlexNet [6], even though it has 50 times fewer parameters. In [7], the proposed architecture, MobileNet, has modified

978-1-6654-2744-9/21/$31.00 ©2021 IEEE

convolutional modules that promote a separable convolution, achieving similar results of a standard convolution with fewer weights. Although these methods achieve competitive results with bigger models, they still rely on human expertise.

Diverse strategies have automated the search for neural network architectures to avoid human effort, such as Reinforcement Learning (RL) [8], EA, and Gradient Descent (GD) [9]. However, RL approaches are hard to train and require a great deal of computational power to achieve their goals [10]. Although GD-based algorithms are more efficient than RL, they demand excessive memory since all candidate networks must be explicitly instantiated [11]. Because of that, in this work, we explore EA strategies.

EA-based approaches have been used for many decades to optimize neural networks architectures. For instance, in 2002, [12] proposed the Neuroevolution With Augmenting Typologies (NEAT), which evolves the nodes and connections of neural networks. Although presenting outstanding results, these early algorithms are not suitable for modern DL models, which have more complex architectures and a more significant number of weights [11]. Recent work puts effort towards the development of solutions to modern architectures. For instance, in [3], the authors proposed a GE approach to find architectures for different goals, such as classification and regression. However, this approach takes much computational time to find effective models. To overcome this problem, in [13] the authors make the algorithm 20 times faster than the previous version without losing accuracy. However, after finding the model, it is necessary to retrain it to readjust the weights. In [4] this problem was tackled by generating neural networks ready for development. Note that these solutions, even with an improvement in time, still need a tremendous amount of GPU power and memory. In other words, they are focused only on achieving the best accuracy without paying much attention to execution time or even the size of the model found. This approach may generate sizeable neural network models, making their use in IoT devices unfeasible.

Resource-constrained devices have been studied in EA-based NAS as well. In [14], the authors proposed NSGA-Net, which used a multi-objective algorithm to not only maximize accuracy but also minimize floating-point operations per second (FLOPS). In [15], the authors modified the tournament selection of the genetic algorithm to favor the younger genotypes. These approaches, however, limit their search space by creating blocks based on well-known DL architectures to accelerate the search at the cost of losing generalization in the search process. Other studies first search neural networks with good accuracy and after compressing them to fit such models to IoT devices with little resources [16], [17]. Although presenting good results, this second step increases their search time.

Inspired by this body of work, our objective is to develop a NAS algorithm that finds models capable of achieving competitive results with large models that are small enough to be implemented in IoT devices. We take advantage of the multi-objective algorithms to produce small and accurate models, and we use GE to codify our network information.

III. Methodology

GEs are evolutionary algorithms that allow the user to incorporate information about the problem in the search space through grammar, defined by a set of terminals, non-terminals, initial symbols, and production rules. We based our method on Dynamic Structured Grammatical Evolution (DSGE) [18], which has a dynamic grammar that allows the evolution of neural network architecture with one or more layers and a diverse number of neurons. We adapt DSGE to account for both small and accurate neural networks, as detailed next.

A. Individual representation

The representation of this work is based on [3]. Each individual represents a DL architecture of two levels, as shown in Figure 1. The first level (the outer level) symbolizes the neural network structure (the sequence of layers); the second level (inner level) depicts the associated parameters of each layer in the first level. Each outer level has a correspondent inner level.

The inner level extracts its parameters from the grammar. Figure 2 shows a sample of the grammar: each production rule contains non-terminals (between angle brackets) and terminals, which in turn contain their possible parameters. For instance, *[rate,float,1,0,0.7]* means that, for the rate parameter, the algorithm must select one float value between 0 and 0.7. By using this approach, it is possible to build several neural network architectures, exploring the search space better than using fixed block architectures, as done in previous studies [16], [17]. Additionally, the user is responsible for defining the values in the grammar; the only step dependent on human expertise in our approach. Thus, there are inherent biases in this choice. However, with extensive grammar, the search space will be large enough to reduce the influence of such bias.

The outer level requires the user to define how many layers of each type (e.g., fully connected or convolutional) can be present in a neural network. This condition limits the maximum structure, preventing the method from generating models that we cannot deploy in resource-constrained devices.

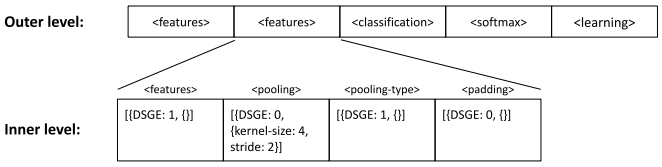

Fig. 1. Individual representation, containing an outer and an inner level (Adapted from [3])

B. Search Space

In this work, we concentrated our efforts on a computational vision problem since it is a common task for IoT devices. Hence, we based our search space on convolutional neural networks (CNNs). In order to develop neural networks capable of preserving accuracy with few parameters, our search space considers the following strategies:

```
<batch-norm> ::= layer:batch-norm
<pooling> ::= <pool-type> [kernel-size,int,1,1,3] [stride,int,1,1,3] <padding>
<pool-type> ::= layer:pool-avg | layer:pool-max
<padding> ::= padding:same | padding: valid
<dropout> ::= layer:dropout [rate,float,1,0.0.7]
```

Fig. 2. A sample of the grammar for the evolution of neural network architectures

- Convolutional kernels of size 1×1, containing nine times fewer parameters than the most used filter of 3×3 [5]. Especially, in the present work, the filters have the option size from 1 to 3, differently from other approaches not focused on resource-constrained devices, such as [3], which have filters from 2 to 5 (and, consequently, more parameters);
- The stride values are from 1 to 3. As described in [5], intuitively, to begin from smaller stride values and increase them in the last layers will create large activation maps, which can lead to higher classification accuracy. However, since our method is automatic, we cannot guarantee that it will choose this strategy.

Table I shows the other neural network parameters that compose our grammar (i.e., our search space) along with their possible values, including their range, if numeric. We chose these values arbitrarily but aiming at the creation of lightweight models. Other parameters are not in the table (namely, bias, nesterov, padding, AMSgrad, and batch normalization) since their values are binary (true or false).

TABLE I
PARAMETERS USED IN THE GRAMMAR.

Parameters		**Values**
Layers	Pooling	average, max
	Dropout	rate ([0, 0.7])
	Convolution	filters ([12, 64]) filter size ([1, 3]) stride ([1, 2])
	Fully-connected	neurons ([20, 128])
Learning	Early stop	[5, 20]
	Activation function	Linear, Tanh, Sigmoid, ReLU
	Batch size	[50, 500]
	Gradient Descent	rate ([0.0001 , 0.1]) momentum ([0.68, 0.99]) decay ([0.000 001, 0.001])
	Adam	rate ([0.0001 , 0.1]) β_1 and β_2 ([0.5, 1]) decay ([0.000 001, 0.001])
	RMSprop	rate ([0.0001 , 0.1]) ρ ([0.5, 1]) decay ([0.000 001, 0.001])

C. Initial population

Note that bigger neural networks do not necessarily mean better classification results, and the use of neural networks of equal size can limit the search space. To avoid such issues, the initial population has neural networks of different sizes. We also restricted the size of each individual in the initial population to start with more superficial structures that can evolve through generations rather than begin with complex structures that may not contribute positively to the search.

D. Fitness

To achieve a lightweight and accurate architecture, we optimize two objectives simultaneously: accuracy and model size (i.e., number of parameters). To this end, we use Non-dominated Sorting Genetic Algorithm II (NSGA-II) [19], an evolutionary multi-objective algorithm based on the concepts of Pareto dominance that finds a set of solutions with the best trade-off between all goals.

Observe that both objectives are conflicting: we want to maximize the classification accuracy while minimizing the model size (which may influence the execution time: the smaller the model, the shorter the execution time). To make implementation simpler, we minimize both classification error and the number of parameters.

We obtain these values of fitness by training the neural network that an individual represents. This training is the bottleneck of the algorithm since the training process may require a lot of computational power and time. To minimize this problem, we use three strategies: (i) early stopping, which monitors the value of the error in a validation dataset and stops the model training when its value stops decreasing; (ii) limited training time, so that after n seconds (a user-defined parameter), the algorithm stops training; (iii) maximum size of parameters, i.e., if the model has more parameters than a user-defined parameter (here 0.5 million), the model is not trained and receives infinity for both objectives.

Hence, after obtaining the fitness for each individual, we randomly select two and compare them in a tournament: the algorithm chooses the individual with the greatest Crowding Distance [19] (i.e., the best trade-off between both fitness) as the parent for the next generation.

E. Genetic operators

After selecting the parent of one generation, we apply genetic operators to create offsprings that will be part of the new generation. Based on [20], we use only mutation operators to speed up the algorithm. This approach is called EA $1+\lambda$, where λ is the number of offsprings generated, and 1 is the parent included in the next generation without changes (an elitist approach). If this parent has not yet reached the maximum number of epochs available, it continues its training in the next generation.

The mutation operations used in this work are the following:

- **Adding hidden layer**: the added layer can be a replication of an existing layer or a new layer, which is added in a random position in the model, respecting the ordering. In other words, the algorithm adds convolutional layers only in the first layers and fully connected layers only in the last layers.
- **Removing hidden layer**: the algorithm chooses a random layer and removes it if this new model does not violate the minimum number of layers. After removing, the algorithm reconnects the anterior and posterior layers.

- **Grammatical mutation**: the algorithm chooses a random layer to randomly change its values by valid values (for instance, integer values are replaced only by integer values, respecting the range determined in the grammar).

IV. Experimental Setup

As previously mentioned, to evaluate the performance of our approach, we focus on a computational vision problem since it is an essential subject in a wide range of IoT applications, such as intelligent transportation, domestic assistants, surveillance, among others [2].

We use the CIFAR-10[1] dataset, which is composed of 32 RGB color images, with 60 000 instances divided in 10 classes. Each class contains 6000 images, 5000 used for training and 1000 used for test. We divide the training set into two disjoint sets: 80 % for adjusting the parameters and 20 % to validate the model, used to evaluate the error (one of our objectives) during the training phase.

We perform the experiments on a machine with the following configuration: Ubuntu 18.04.3 OS, 20 × Intel(R) Core(TM) i9-9900X CPU @ 3.50 GHz, 128 GB RAM and a GPU Nvidia RTX 6000. We implement all methods in Python (version 3.7.3). Specifically, we implemented the neural networks in Keras[2] (version 2.4.3), which executes the neural network models in GPU.

Since EAs are stochastic, we execute the experiments five times. For the EA, we use a 4 λ offsprings + 1 parent strategy, evolved for 50 generations. For mutation, the probabilities of adding or reusing a layer were equal to 15%; removing a layer, 30%; and of grammatical mutation, was 20%. The evolved architectures were allowed a maximum of 10 CNN layers and 10 FC layers. We choose these values arbitrarily, having in mind the assemble of lightweight models for resource-constrained IoT devices. Regarding the parameters of the neural network not specified by the grammar, we use *softmax* as the activation function of the last layer, with categorical cross-entropy as the loss function. The number of epochs, in contrast with other approaches in the literature that train the model for very few epochs during evaluation, such as 10 [3] or 25 epochs [14] – which can prevent an adequate adjustment of the neural network parameters, we train our networks for 500 epochs. Since our approach aims at small models, we can train them for more epochs, which can help find better models. The maximum training time allowed for a model is 300 seconds.

V. Experimental Results and discussion

Figure 3 shows the classification accuracy, the number of parameters, and the number of hidden layers of the best individual in each generation. The values are an average of the five independent executions.

Note that the accuracy does not decrease during evolution. It happens thanks to elitism, which keeps the best individual in each generation in the population. We observe that the accuracy increases in the first generations, but close to the 20th generation, it stabilizes for about ten generations. This fact happens again close to the 35th generation. A hypothesis to explain this phenomenon is the sharp exploration profile of our proposal: it does not have explicit exploitation, such as the crossover operation, focusing on exploring the search space through mutation. Hence, it is "easier" to find regions in the search space that present good results initially, but without concentrating in the neighborhood of the best individuals. However, this behavior may prevent the algorithm from being stuck into the local maximum search space.

Regarding the number of parameters, as the accuracy stops increasing, the number of parameters stops – possibly because the same individual is maintained over a few generations until the algorithm finds a better one. Between the 30th and 40th generations, the accuracy shows a slight improvement, while the number of parameters remains approximately the same. Note that the produced models are small, and none exceeds 2 megabytes (MB), which we may incorporate into several IoT objects with memory restrictions.

Furthermore, the number of hidden layers increases as the generations evolves. However, the average size is still tiny: the best individual in the 50th generation contains about seven hidden layers. When comparing these results with the ones shown in the previous graphs, we realize that the proposed method improves model performance during the search and increases the number of layers while decreasing their complexity (i.e., we have the same number of parameters in more hidden layers).

Table II presents the model accuracy on the validation set, the number of hidden layers, and the number of parameters obtained by the best individuals in each replication. The 3rd replication has the individual with the best accuracy, with 81.00 % and the fourth replication has the individual with the worst accuracy, 66.25 %. The other replications obtained individuals that reached between 70 % and 75 % of accuracy. Although the best individual in the 3rd replication has the most significant number of hidden layers, the individual in the 4th replication has the most significant number of parameters by far – and the worst result, which suggests overfitting. Note that more layers do not imply better accuracy or more parameters; similarly, more parameters do not imply better results or more layers.

TABLE II
Results for the best individuals for each execution.

Execution	Accuracy	Hidden layers	Parameters
1	71.05 %	5	23 593
2	74.71 %	6	74 829
3	81.00 %	12	153 062
4	66.25 %	6	440 550
5	75.34 %	5	75 872

Figure 4 shows the architecture of the best individual obtained in the 3rd execution, which achieved the best result in the validation set. The right side shows the output of each layer and, on the left side, the configuration of each layer. The last convolutional layers increase the size of the convolutional

[1]https://www.cs.toronto.edu/ kriz/cifar.html
[2]https://keras.io/

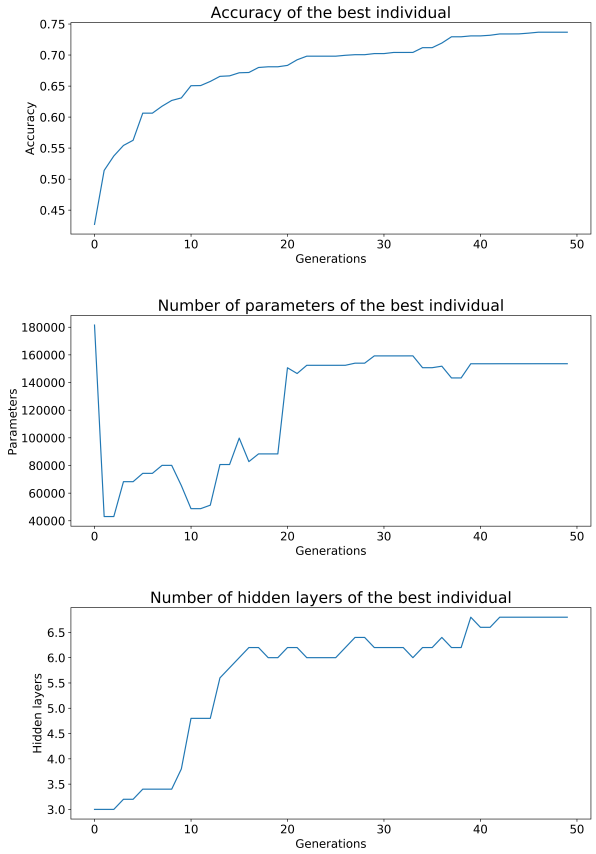

Fig. 3. Evolution of the accuracy values (first plot) and the number of parameters (second plot), which are the evaluated fitness. The third plot shows the number of hidden layers of the best individual obtained over the generations. These results are the average of five independent replications.

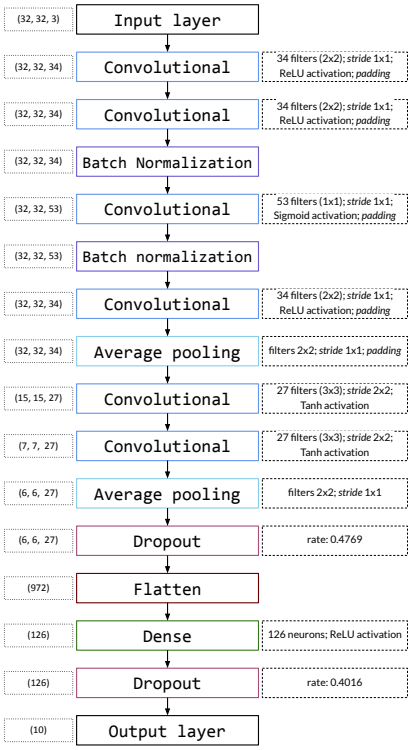

Fig. 4. Best individual obtained in the validation set

filter and the stride. As said in Section III, this strategy increases the activation maps, leading to an improvement in the classification results. This result is interesting because our proposal automatically evolved to something previously theorized in [5].

Moreover, we note that the model uses three activation functions (Tanh, ReLU, and Sigmoid), which is uncommon in the literature. We can also observe that the input size decreases four times, starting at 32×32 and ending at 6×6 since it does not use padding in the three layers before dropout. These choices should be better investigated in future studies to understand their benefits.

Table III depicts a comparative analysis of the accuracy on the test set of CIFAR-10, along with the number of parameters (in million), and highlights the following features of the presented techniques: NAS approach (an advantage, since it does not depend on human expertise), lightweight (it generates small models that we can deploy to IoT devices), and blocks (a disadvantage since this kind of search depends on human expertise to build blocks based on well-known DL models). SqueezeNet[3] [5] is a hand-crafted approach focused on lightweight models. The other approaches use NAS. However, DENSER[4] [3] focuses only on classification results, without accounting for model size. DeepMaker [17] and NSGA-Net [14] are both NAS approaches focused on lightweight models, but they use blocks from well-known DL architectures designed by human experts in their search space. Hence, their search is biased. DeepMaker first executes its NAS strategy, finding an architecture, and prunes it through a magnitude-based approach (i.e., prunes the weights and retrains the architecture, which can be time-consuming).

TABLE III
COMPARISON WITH OTHER STUDIES

	Acc.	Param. (M)	NAS	lightweight	blocks
Our proposal	80.31 %	0.15 ∼2 MB	✓	✓	
DENSER	88.41 %	∼84 MB	✓		
SqueezeNet	80.00 %	0.12		✓	
DeepMaker	85.9 %	0.14	✓	✓	✓
NSGA-Net	96.15 %	3.3	✓	✓	✓

It was impossible to execute the DENSER models available since they use old libraries of Python version 2.7. Although we do not know the number of parameters of the model, we know models with thousands of parameters as ours have a size of about 2 MB. Hence, we can infer that the model found by DENSER, which is about 40 times bigger than ours, contains millions of parameters. Even though the model found

[3]https://github.com/zshancock/SqueezeNet_vs_CIFAR10

[4]https://github.com/fillassuncao/denser-models

by DENSER achieves higher accuracy than ours, its size of $84\,\mathrm{MB}$ prevents its use in strict resource-constrained devices.

The NAS lightweight approaches, DeepMaker and NSGA-Net, also achieves better accuracy results than our approach. However, they present main drawbacks when compared to ours: both strategies use blocks based on DL well-known architectures in their search space, not only inserting biases but also needing more human knowledge to develop such blocks manually. Our approach builds neural networks from scratch based on the hyperparameters described in the grammar; consequently, our approach needs less human intervention. The model found by the NSGA-Net contains 3.3 millions of parameters (about $20\,\mathrm{MB}$), which also prevents its use in strict resource-constrained IoT objects. The DeepMaker has a pruning module; that is, they perform the neural architecture search, and after finding an acceptable model, they prune its weights. Our approach does need any compression technique.

Our results are similar in accuracy and number of parameters to those achieved by the SqueezeNet. Additionally, our model automatically developed something similar to their strategy to maximize accuracy with a limited number of parameters, which decreases the convolutional filter size and after expands it to create larger activation maps and thus increases the accuracy. The authors manually developed this architecture (i.e., it needs human expertise), while our model automatically achieved its results.

Finally, we retrained the found model in the MNIST[5] dataset to evaluate the capability of generalization of the models found by our proposal. We achieved $99.05\,\%$ in the validation set and $99.18\,\%$ in the test set, showing shreds of evidence that support that our model can generalize.

VI. Final Remarks

We developed a NAS strategy based on Structured Grammatical Evolution and NSGA-II. Using model size and accuracy as fitness, our proposal was able to generate a Convolutional Neural Network model with less than $2\,\mathrm{MB}$, achieving an accuracy of about $81\,\%$ in the CIFAR-10 dataset and $99\,\%$ in MNIST, with only 150 thousand parameters approximately. Hence, our approach can be deployed in very resource-constrained devices, as we can commonly see in IoT.

As future steps, we envision investigating how better controlling the exploitation could influence the results through the crossover operation. We also intend to study different techniques focused on decentralized contexts, as the IoT scenarios frequently are.

Acknowledgment

This work was partially funded by CNPq (grant 311750/2018-4), CAPES, FAPEMIG, and grant #2020/05121-4, São Paulo Research Foundation (FAPESP).

[5]http://yann.lecun.com/exdb/mnist/

References

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[2] D. Trihinas, G. Pallis, and M. Dikaiakos, "Low-cost adaptive monitoring techniques for the internet of things," *IEEE Transactions on Services Computing*, 2018.

[3] F. Assunçao, N. Lourenço, P. Machado, and B. Ribeiro, "DENSER: deep evolutionary network structured representation," *Genetic Programming and Evolvable Machines*, vol. 20, no. 1, pp. 5–35, 2019.

[4] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Fast-DENSER++: Evolving fully-trained deep artificial neural networks," *arXiv preprint arXiv:1905.02969*, 2019.

[5] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[8] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[9] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," *arXiv preprint arXiv:1711.04528*, 2017.

[10] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. Yen, "A survey on evolutionary neural architecture search," *arXiv preprint arXiv:2008.10937*, 2020.

[11] H. Zhu, H. Zhang, and Y. Jin, "From federated learning to federated neural architecture search: a survey," *Complex & Intelligent Systems*, vol. 7, no. 2, pp. 639–657, 2021.

[12] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[13] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Fast DENSER: Efficient deep neuroevolution," in *EuroGP*, 2019, pp. 197–212.

[14] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 419–427.

[15] E. Real, A. Aggarwal, Y. Huang, and Q. Le, "Regularized evolution for image classifier architecture search," in *Proc of the AAAI*, vol. 33, no. 01, 2019, pp. 4780–4789.

[16] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," *arXiv preprint arXiv:1911.00105*, 2019.

[17] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin, "DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocessors and Microsystems*, vol. 73, p. 102989, 2020.

[18] N. Lourenço, F. Assunção, F. B. Pereira, E. Costa, and P. Machado, "Structured grammatical evolution: a dynamic approach," in *Handbook of Grammatical Evolution*. Springer, 2018, pp. 137–161.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[20] E. Real, C. Liang, D. R. So, and Q. V. Le, "AutoML-Zero: Evolving machine learning algorithms from scratch," *arXiv preprint arXiv:2003.03384*, 2020.