Q-NAS revisited: exploring evolution fitness to improve efficiency

Daniela Szwarcman
PUC-Rio, IBM Research
Rio de Janeiro, Brazil
daniela.szw@ibm.com

Daniel Civitarese
IBM Research
Rio de Janeiro, Brazil
sallesd@br.ibm.com

Marley Vellasco
Dept. of Electrical Engineering
PUC-Rio
Rio de Janeiro, Brazil
marley@ele.puc-rio.br

***Abstract*—Over the last decade, the scientific community has witnessed the success of deep neural networks in a variety of tasks. However, the design of these new structures still demands expert knowledge and significant time. In this scenario, the idea of automating the design of such complex networks has inspired many researchers. New algorithms have been proposed to address the neural architecture search problem, but computational cost is a major drawback. Q-NAS (Quantum-inspired Neural Architecture Search) is an evolutionary algorithm recently proposed with the idea of improving efficiency, but with minor human-bias introduced in the search. This work extends the analysis of Q-NAS, focusing on fitness behavior during evolution. We want to verify if good individuals are obtained early in evolution, so we could apply an early-stopping mechanism. We experiment with a simple early-stopping technique, and results indicated an evolution time reduction of more than 45% in most cases. This new feature can improve efficiency, making Q-NAS very competitive compared to other algorithms.**

***Index Terms*—Neural architecture search, evolutionary algorithms, deep learning, quantum inspired algorithms**

I. Introduction

The recent success of convolutional neural networks (CNN) for image applications is confirmed by the great performance of architectures such as AlexNet [1], VGGNet [2], and ResNet [3]. These networks were hand-designed, and their overall structure is still comparable to the first practical CNN, the LeNet model [4].

Neural Architecture Search (NAS) is the process of automating the design of neural networks. NAS is a growing area of interest in the machine learning community, as the design of deep systems requires expert knowledge and considerable time [5]. To address the NAS problem, researchers have proposed algorithms based on different techniques, including reinforcement learning (RL) [6]–[8] and evolutionary algorithms (EA) [9], [10]. Several approaches present competitive results compared to state-of-the-art hand-made architectures. However, many of them require significant computational resources [6], [9]–[11].

More recently, new solutions to address NAS considering the computational cost problem have emerged. The new ideas include the use of network transformations [12], early stopping techniques [8], block search [8], [13] among others. In most of them, the trade-off to improve efficiency is the addition of human-bias to reduce the search space. Therefore, new algorithms that can balance efficiency and generality represent a great advance for the NAS current state-of-the-art.

Considering the importance of this balance, the authors in [14] presented Q-NAS (Quantum-inspired Neural Architecture Search), which applies a quantum-inspired evolutionary algorithm (QIEA) to the NAS problem. QIEAs are a category of evolutionary methods which uses ideas from quantum computing, such as superposition of states [15], [16]. When compared to related algorithms, empirical studies have confirmed that QIEAs can find better results with fewer evaluations for many optimization problems [17], [18]. The authors of Q-NAS claim it is a promising new step towards the efficiency of NAS with minor human bias [14]. They reported a final network with less than 20 layers and 89% of accuracy in the CIFAR-10 dataset, using 20 K80 GPUs for about two days.

In this work, we extend the analysis presented in [14], focusing on the analysis of the fitness function during the evolutionary process. The idea is to verify if Q-NAS can benefit from an early-stopping mechanism. If the final networks are found relatively early during evolution, Q-NAS would gain in efficiency with a mechanism that could identify stagnation and stop the algorithm sooner. Furthermore, we experiment with elitism selection, which is less conservative than the steady-state used in [14]. We want to check if a more diverse population can help to find better structures faster.

This paper has four additional sections, organized as follows. Section II presents an overview of relevant NAS works; the Q-NAS algorithm is described in Section III; Section IV speficies the experiments and discuss the results; lastly, Section V presents the conclusions.

II. Related Work

Automatic design of neural networks has been investigated before, as confirmed by the extensive work on neuroevolution already published. These methods apply EAs to evolve the network weights as well as its structure [19], [20]. However, the former neuroevolutionary approaches were restricted to smaller networks and did not scale to the new deep structures. Recent NAS study focuses exactly on the automatic engineering of deep complex networks.

One important work on NAS was presented by Zoph and Le [6] a few years ago, showing competitive results with hand-designed architectures. The authors applied a reinforcement

2643-6264/19/$31.00 ©2019 IEEE
DOI 10.1109/BRACIS.2019.00095

learning technique to search for well-performing deep models. They generate variable length strings to encode the network structure using a recurrent neural network as the controller in the RL system. In the case of image classification, the generated structures are trained on the complete CIFAR-10 [21], and the accuracy is the reward in the reinforcement learning system. Although their results were relevant and stimulated the NAS research, they used 800 GPUs for more than 3 weeks to achieve them.

Evolutionary techniques have been applied to NAS likewise, with the evolution occurring only in the structure space and weights being adjusted by regular gradient-based training [5]. In the evolutionary method proposed in [9], the authors use a one-shot technique, which produces a fully-trained network at the end of the search. The method starts with networks of only one layer and mutation operators allow them to grow. They used 250 GPUs for 256 hours to search for their best network, claiming their search space is wide.

The high computational cost of NAS methods inspired scientists to create new algorithms aiming to improve efficiency. Istrate et al. [22] introduce an accuracy predictor that estimates the classification performance of a network without training it. They achieve promising results, especially regarding efficiency. However, their method relies on an external database of known datasets and a metric to rank them by difficulty.

A recurrent idea to improve efficiency is to limit the search space. For example, optimize blocks, or motifs instead of complete structures [10], [11], [13]. The network is formed by stacking the candidate blocks in a pre-defined way, determining the so-called meta-architectures. Even with the strong human-bias to limit the search space [5], the computational resources are still expressive: Liu et al. [13] used 200 GPUs for 1.5 days to achieve their best result on CIFAR-10; Zoph et al. [11] used 4 days with 500 GPUs; and Real et al. [10] ran experiments for 7 days with 450 GPUs.

III. Q-NAS ALGORITHM

A. Quantum-inspired evolutionary algorithms

Quantum-inspired computing takes advantage of the principles of quantum physics to create algorithms that can be executed on classical computers [23], [24]. A QIEA applies these principles to solve optimization problems. Similar to other evolutionary algorithms, a QIEA is characterized by a population of individuals that represents possible solutions to the problem. A canonical EA can be summarized as follows: (1) generate initial population; (2) evaluate and rank individuals; (3) modify population using recombination or mutation; (4) select individuals for the next population; (5) repeat [2, 3, 4] until a stopping condition is reached.

A key difference between QIEAs and other EAs is the quantum population, which represents a superposition of states (possible solutions) in the search space [16]. Note that, since a quantum individual encodes many solutions, it can only be evaluated when it collapses to a single state, generating a classical individual. The observation process to create the classical population depends on the chosen quantum representation.

The quantum individual can be represented by a string of q-bits, for example. The state $|\Psi\rangle$ of a single Q-bit $[\alpha \quad \beta]^T$ is defined as [15], [16]:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

where α and β are complex numbers. The Q-bit collapses to the state "0" with probability $|\alpha|^2$ and to the state "1" with probability $|\beta|^2$. When the evolution begins, the Q-bit individual represents all possible states in the search space with the same probability. During evolution, quantum operators can modify the probabilities in each q-bit, so it can gradually converge to a single state that should represent the optimal solution for the variable it is encoding.

QIEAs with other quantum representations have been introduced, such as the numerical representation for real variables [17], [18], and a combination of binary and numerical, for tasks with categorical and real variables [20], [25].

B. Q-NAS steps

Q-NAS is a quantum-inspired evolutionary algorithm focused on the deep neural architecture search problem and currently framed in the context of image classification tasks [14]. We first present an overview of the Q-NAS algorithm; the details of the quantum population and candidate networks representation will be reviewed later on this section.

Figure 1 shows the Q-NAS summarized steps. The algorithm starts with the initialization of the quantum population $Q(t)$ (line 2), which consists of assigning initial probability values to each quantum individual.

```
 1: t ← 0
 2: Initialize Q(t)
 3: while t ≤ T do
 4:     Generate classical population C(t) observing Q(t)
 5:     if t = 0 then
 6:         Evaluate C(t)
 7:         P(t) ← C(t)
 8:     else
 9:         C(t) ← recombination between C(t) and P(t)
10:         Evaluate C(t)
11:         P(t) ← select(C(t), P(t))
12:     end if
13:     Q(t + 1) ← update Q(t) based on P(t) values
14:     t ← t + 1
15: end while
```

Fig. 1: Q-NAS algorithm.

At the beginning of the loop of generations, $Q(t)$ is observed to generate the classical population $C(t)$ (line 4 in Figure 1). Each quantum individual can produce more than one classical individual. Once ready, $C(t)$ can be evaluated (line 6 in Figure 1). The evaluation procedure involves training the candidate networks for a small number of epochs with a subset of the training data and using a validation dataset to assign a fitness score to the individual. In the first generation, $C(t)$ individuals are ranked and stored in $P(t)$ (line 7 in Figure 1). Note that classical recombination is only possible after the first generation (Figure 1, line 9).

In the subsequent generations, we already have a ranked $P(t)$ population, so when we generate new classical individuals we must select which ones will stay in $P(t)$ and which ones will be replaced by new ones (line 11 in Figure 1). In this paper, we will analyze two selection mechanisms: steady-state and elitism. The first method orders the old and new population and keeps the k best individuals; the second replaces all the old individuals except the best one.

Completing the loop, $Q(t)$ is updated based on the best individuals in $P(t)$ (Figure 1, line 13). The idea is to gradually modify the quantum population so it can generate solutions that are closer to the optimal. In other words, the update should reduce the search space and also map promising search areas.

The loop terminates when a maximum number of generations T is reached. After the evolution is complete, the best individual represents the final architecture description. This network is then retrained with the entire training dataset and the test set is used to assign a final performance metric – accuracy, in this case – to the structure. This metric value is used to compare the performance of Q-NAS generated models with other works [14].

C. Q-NAS population

Q-NAS aims to address not only the network design but also the optimization of some hyperparameters related to the training procedure [14]. Therefore, Q-NAS must be able to represent the categorical space of the network structures and the numerical space of the training hyperparameters.

The network architecture is encoded as a chain-like structure with a maximum size L nodes. In this structure, each node has a layer (or block of layers) function associated with it. The user specifies a list of predefined functions which will be the search space for every node; the names of the functions are mapped to integers. The authors in [14] include a "no operation" (*NoOp*) function in the list, so they can represent variable length structures. The network part of the classical chromosome is simply an array of integers representing the functions associated with each node [14].

The quantum individual will define a unique probability mass function (PMF) for each node. Considering N quantum individuals, a maximum network size L and a function list of size M, the quantum population will be an array of shape (N, L, M). The initial PMF is the same for all nodes, but the probability values for each function can be defined by the user. This enables the user to increase the initial probability of some functions over the others if this kind of bias is desired.

The observation process to obtain the classical individuals consists of sampling from the PMF of each node separately. Numpy [26] discrete sampling is used. Figure 2 shows an example of this process with $L = 3$ and $M = 5$.

In line 13 of Figure 1, the quantum population is updated based on the best classical individuals. For the network chromosome part, a simple heuristic is presented in [14]. The idea is to increase the probability of a promising function in a node and decrease proportionally the other probabilities.

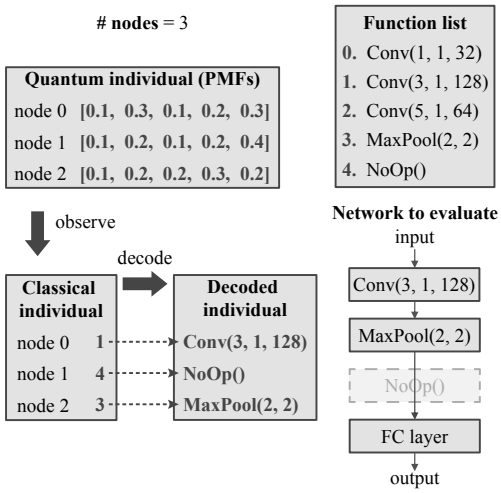

Fig. 2: Network generation procedure. *Conv(*k*,* s*,* f*)* is a convolution layer with kernel size $k \times k$, stride s and f filters. The observation process samples from the PMFs of each node. The decoding procedure maps integers to function names. The final network includes a fully connected (FC) classifier layer.

The second part of the chromosome uses a numerical representation based on [17], [18] to encode some training hyperparameters. For each variable h_j to be optimized, a probability density function (PDF) is defined, respecting its domain. Considering a set of G numerical hyperparameters, the quantum chromosome q_i is an array in the form [14]:

$$[g_{i1} = p_{i1}(x), ..., g_{iG} = p_{iG}(x)] \tag{2}$$

where $p_{ij}(x)$ is the PDF that represents the probability of observing a quantum gene in a particular range of values when it is observed. In [14], the authors use a uniform PDF, which is defined by its lower and upper limits l_{ij} and u_{ij} – the minimum and maximum values h_j can assume.

Once again, the observation process consists of sampling from the distributions. In the case of the uniform PDFs, the sampling procedure is straightforward and uses its corresponding CDF $F_{ij}(x)$. It involves the following steps: (1) generate a random number r in the interval [0, 1]; (2) find the x such that $F_{ij}(x) = r$; this is the sampled value. As $F_{ij}(x)$ is a straight line inside the domain, x can be obtained by [14]:

$$x = r \times (u_{ij} - l_{ij}) + l_{ij} \tag{3}$$

Exclusively in the numerical part of the classical population, the authors in [14] apply traditional recombination (line 9 in Figure 1), more specifically the arithmetic crossover operator.

The quantum update for the numerical part of the chromosome is a heuristic that also uses values from the best classical individuals. This heuristic shifts the PDFs in the direction of the best values and reduces their width according to the range of current classical values. Details about this update can be found in [14].

IV. Experiments and discussion

The first Q-NAS results presented in [14] are the starting point of our new experiments. We want to analyze the population fitness throughout the evolution process with two main goals. The first one is to compare the steady-state selection mechanism used in [14] with the less conservative elitism selection. The second one verifies if Q-NAS can find good individuals early in evolution and if it could benefit from the addition of an early-stopping mechanism.

A. Experiment 1

The first experiment involves comparing the population fitness of runs presented in [14] with the new runs using elitism rather than steady state. We use the same parameter configurations of [14], and we will repeat them here for clarity.

The evaluation step in Q-NAS comprises training and validation of the candidate networks using a subset of the available data, as mentioned in Section III. We used the same CIFAR-10 [21] dataset scheme presented in [14], detailed as follows. For the candidate evaluation, a subset of the original CIFAR-10 training set was used to create both training and validation sets. We randomly selected 900 examples per class for training and 100 per class for validation, leading to a total of 9000 images for training and 1000 for validation. Mean subtraction was the only preprocessing method. For data augmentation, we used random flip and zero-padding combined with random crop.

The RMSProp optimizer is used in the training procedure [14], which has two hyperparameters called *decay* and *momentum* [27]. In [14], two different configurations for these hyperparameters were used, which we will refer here as *config 1* and *config 2*. The first one uses fixed values for both decay and momentum, as seen on the right side of Table I. In *config 2*, these hyperparameters were evolved by Q-NAS along with the weight decay hyperparameter, using the numerical part of the chromosome. In summary, the difference between configurations relies only on the evolution of hyperparameters. In *config 1*, no hyperparameters are evolved, only the network structure, while in *config 2* three hyperparameters are evolved together with the network structure. For Experiment 1, we used only *config 1*, as the best result in [14] occurred for this case.

Table I (left) also shows the values for the Q-NAS specific parameters selected in [14] and kept equal here. The number of nodes is fixed as 20, which means that the networks can have at most 20 layers. The repetition factor is the number of classical individuals each quantum individual will generate – 4 in this case. The *update generations* parameter specifies the frequency in which the quantum update procedure will be executed with a rate defined by the *update quantum rate* parameter. Based on the values in Table I, the quantum update will take place every 5 generations, with a rate of 0.1.

In Section III, we remark that the Q-NAS user must specify the functions that will compose the search space for each node in the network. Table II shows the selected functions, which we can divide in three types: *ConvBlock*, *Pooling*, and *NoOp*. The *ConvBlock* comprises a convolutional layer, a batch normalization layer, and ReLU activation. The specific kernel sizes, strides, and the number of filters are listed in Table II. For the *Pooling* layer, we have two options: max pooling and average pooling, both with a kernel size of 2 and stride of 2. The function specifications favor simplicity and computational cost efficiency [14]. Notice that the convolutional layers do not reduce the input size; the size reduction is left to the pooling layers only. As already mentioned, the *NoOp* function is added so we can represent networks of variable effective lengths. Table II also provides the initial probabilities assigned to each function, which are equally divided by function type (a total of $1/3$ for each one).

TABLE I: Q-NAS parameters for the experiments

evolution parameters		numerical hyperparams	config 1	config 2
generations	300	decay	0.9	[0.1, 0.999]
# network nodes	20			
# quantum ind.	5	momentum	0.0	[0.0, 0.999]
repetition	4			
update generations	5	weight decay	1.0e-4	[1.0e-5, 1.0e-3]
update quantum rate	0.1	learning rate	1.0e-3	1.0e-3

TABLE II: Layer functions and initial probabilities

function type	kernel size	stride	filters	op.	initial probability
ConvBlock	1	1	[32, 64]	-	0.042, each option
	3	1	[32, 64, 128, 256]	-	
	5	1	[32, 64]	-	
Pooling	2	2	-	[max, avg]	0.167, each option
NoOp	-	-	-	-	0.333

We repeat five evolution runs using the specified parameters and the *config 1* hyperparameters. Each candidate network is evaluated in a K80 GPU, summing up to 20 GPUs (same as in [14]). The networks are trained for 50 epochs, with a batch size of 256; the validation accuracy is calculated for the last 5 epochs. For the retraining phase, we use all the available training data and two K80 GPUs, for 300 epochs. The final accuracy is obtained using the test set.

Table III shows the outcomes for Experiment 1, where we detail the final network size and its accuracy on the test set after the retraining phase. On the left, we repeated the values for *config 1* from [14]. The right side of Table III shows the new runs with elitism selection. Notice that in terms of final accuracy, the selection method does not seem to affect the final accuracy values. It should be noted that the network from steady-state#4 presented a bad result, but its accuracy in the evolution phase was 0.79 [14]. This means that this particular 10 layer network does not have enough capacity [14]. However, the evolution times for the elitism case are longer. This may indicate that the steady-state method

is guiding the evolution in the direction of faster training structures, which are not always the best performing ones.

TABLE III: Results for Experiment 1

	steady-state			elitism		
#	evolution time	# layers	retrain acc.	evolution time	# layers	retrain acc.
1	56h	13	0.8773	64h	14	**0.8957**
2	52h	11	0.8812	60h	10	0.8767
3	61h	12	**0.8928**	64h	14	0.8872
4	54h	10	0.4925	62h	15	0.8876
5	56h	15	0.8874	63h	14	0.8833

Figure 3 shows the best individual fitness and the average population fitness during evolution, for the best runs of steady-state and elitism. The difference between the two methods becomes more evident if we compare the average population fitness. Although the average grows quite slowly, and the standard deviation is small in the steady-state case, there is still increase in the best fitness value. In the elitism case, the best fitness curve presents similar characteristics, even though the population fitness average behaves very differently.

The plateau nature of the best fitness curve indicates that it is challenging to find significantly better structures. The average fitness for the elitism case corroborates with this idea. In this scenario of greater diversity (all individuals are replaced except the best one), several bad performing structures are generated. In other words, the population is not suffering from stagnation considering the network structures; different architectures present equally bad results, and it is difficult to find one with a stand out performance. This idea could also explain the slow increase in the average fitness for the steady-state case: since only the best ones are selected, it is hard for the new individuals to get into the population. In summary, the elitism runs helped us understand the evolution process, but we did not see a clear advantage of using this method rather than the steady-state.

B. Experiment 2

After analyzing the fitness characteristics in Experiment 1, we defined a simple early-stopping criterion: *stop if the best individual fitness does not improve above a threshold of 0.005 for 80 generations*. The second experiment involves retraining the best individual at the generation indicated by this early-stopping criterion. We should highlight that we chose the values for threshold and generations based on the behavior we observed in the experiments so far. Our intention here is only to verify the applicability of a simple early-stopping method in Q-NAS.

Since we have the population descriptions and fitness values saved for the evolution runs, it is possible to apply the early-stopping criterion on the saved data. This method allows us to compare the final network and the early-stopping one from the same run, which reduces the influence of random seeds and initialization issues. We applied the criterion to all the runs in [14] and retrained the indicated models.

In Table IV, we repeated the results for *config 1* and *2* from [14] on the left, including the final accuracy, evolution time and network size. On the right side of Table IV, we present the results for the last models indicated by the early-stopping criterion of Section IV. The *gen* column on the right shows the number of generations Q-NAS would have run if it was terminated by the early-stopping method. We used timestamps in our log files to estimate the evolution time for these cases. In some lines of Table IV, the retrain accuracy is unchanged with the addition of early-stopping, which means that it stopped when the final model was already found. This observation corroborates with the idea that Q-NAS may find good structures early in the evolution process.

In all runs, there is a significant time reduction, with a remarkable decrease of 75% in the best case (run 2-3). Regard-

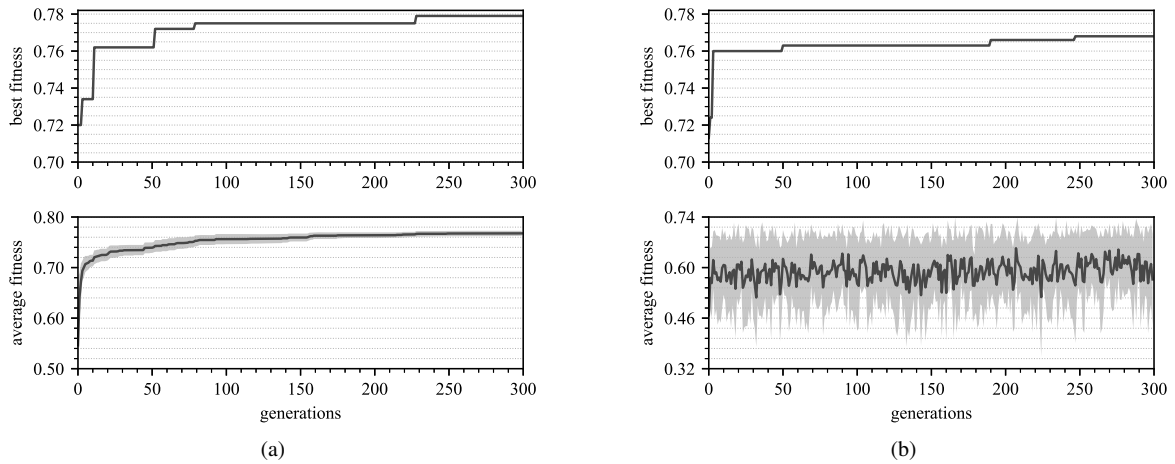

Fig. 3: Best individual fitness (top) and average population fitness, along with the standard deviation (bottom) for (a) steady-state selection, run #3 and (b) elitism selection, run #1.

TABLE IV: Results for Experiment 2

	no early-stopping				with early-stopping			
config -run	gen.	evol. time	net size	retrain acc.	gen.	evol. time	net size	retrain acc.
1-1	300	56h	13	0.8773	162	29h	13	0.8773
1-2	300	52h	11	0.8812	88	15h	12	0.8867
1-3	300	61h	12	**0.8928**	133	25h	12	**0.8912**
1-4	300	54h	10	0.4925	162	28h	10	0.4925
1-5	300	56h	15	0.8874	174	31h	15	0.8874
2-1	300	52h	12	0.8765	90	15h	10	0.8362
2-2	300	52h	11	0.8620	165	28h	11	0.8620
2-3	300	54h	16	**0.8805**	82	13h	16	**0.8805**
2-4	300	51h	15	0.8575	86	14h	15	0.8575
2-5	300	51h	11	0.8430	163	27h	17	0.8346

*gen. = generations; net size = number of final network layers

ing the final network performance, the method is adequate, as all models on the right (with early-stopping) are equally or almost equally good as the models on the left (no early-stopping). However, further investigation is needed to verify if the mechanism is applicable to other datasets. Furthermore, it is important to check if the threshold and generations in the criterion are robust choices; otherwise, this could invalidate the method.

V. Conclusions

In this work, we extended the previous analysis of the Q-NAS algorithm, focusing on the evaluation of the fitness function behavior during the evolutionary process. We experimented with the elitism selection mechanism that increases the population diversity. This new selection mechanism did not provide better-performing structures in a faster way.

We observed a plateau behavior in the best fitness curves that inspired us to test a simple early-stopping method, which considers only the changes in these values. The application of the early-stopping mechanism not only reduced the evolution time but maintained the quality of the final individuals. In some cases, we were able to stop at a generation that contained the final best model, spending 1/4 of the original time. Our results show that the addition of this simple technique can improve Q-NAS efficiency significantly. Future work involves testing the same early-stopping mechanism in other datasets in order to confirm this expectation.

Acknowledgment

The authors would like to thank the Brazilian Agencies CNPq and FAPERJ for their financial support.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Sep 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, December 2015.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[5] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2018, in press, available at http://automl.org/book.

[6] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *arXiv:1611.01578 [cs]*, November 2016.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning," *arXiv:1611.02167 [cs]*, November 2016.

[8] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. Liu, "Practical Block-Wise Neural Network Architecture Generation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 2423–2432.

[9] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. Sydney, Australia: PMLR, June 2017, pp. 2902–2911.

[10] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," *arXiv:1802.01548 [cs]*, Febuary 2018.

[11] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 8697–8710.

[12] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-Level Network Transformation for Efficient Architecture Search," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[13] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical Representations for Efficient Architecture Search," in *International Conference on Learning Representations*, 2018.

[14] D. Szwarcman, D. Civitarese, and M. Vellasco, "Quantum-Inspired Neural Architecture Search," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, accepted for publication.

[15] M. D. Platel, S. Schliebs, and N. Kasabov, "Quantum-Inspired Evolutionary Algorithm: A Multimodel EDA," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 6, pp. 1218–1232, December 2009.

[16] K. Han and J. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 6, pp. 580–593, December 2002.

[17] A. V. A. da Cruz, M. Vellasco, and M. Pacheco, *Quantum-Inspired Evolutionary Algorithm for Numerical Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 19–37.

[18] ——, "Quantum-inspired evolutionary algorithms applied to numerical optimization problems," in *IEEE Congress on Evolutionary Computation*, July 2010, pp. 1–6.

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

[20] M. Vellasco, A. V. A. Cruz, and A. Pinho, "Quantum-inspired evolutionary algorithms applied to neural modeling," *IEEE World Conference on Computational Intelligence, Plenary and Invited Lectures*, pp. 125–150, 2010.

[21] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Tech. Rep., 2009.

[22] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, and A. C. I. Malossi, "TAPAS: Train-less Accuracy Predictor for Architecture Search," *arXiv:1806.00250 [cs]*, June 2018.

[23] M. Moore and A. Narayanan, "Quantum-inspired computing," Dept. of Computer Science, University of Exeter, Exeter, Tech. Rep., 1995.

[24] A. Narayanan and M. Moore, "Quantum-inspired genetic algorithms," in *Proceedings of IEEE International Conference on Evolutionary Computation*, May 1996, pp. 61–66.

[25] M. C. Cardoso, M. Silva, M. M. B. R. Vellasco, and E. Cataldo, "Quantum-Inspired Features and Parameter Optimization of Spiking Neural Networks for a Case Study from Atmospheric," *Procedia Computer Science*, vol. 53, pp. 74–81, 2015.

[26] O. Travis E, "A guide to numpy," USA, 2006.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.