Julia Drummond Noce

Enhanced Q-NAS for Image Classification

Dissertação de Mestrado

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica, do Departamento de Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica.

Advisor : Prof. Marley Maria Bernardes Rebuzzi Vellasco
Co-advisor: Prof. Karla Tereza Figueiredo Leite

Rio de Janeiro
April 2022

Julia Drummond Noce

Enhanced Q-NAS for Image Classification

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica. Approved by the Examination Committee:

Prof. Marley Maria Bernardes Rebuzzi Vellasco
Advisor
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Karla Tereza Figueiredo Leite
Co-Advisor
UERJ

Dr. Daniela De Mattos Szwarcman
IBM Research

Prof. Douglas Mota Dias
UERJ

Prof. Celso Gonçalves Camilo Junior
UFG

Rio de Janeiro, April 20th, 2022

Julia Drummond Noce

Graduated in Computer Science at the Universidade Federal Fluminense in 2019.

Bibliographic data

Noce, Julia

Enhanced Q-NAS for Image Classification / Julia Drummond Noce; advisor: Marley Maria Bernardes Rebuzzi Vellasco; co-advisor: Karla Tereza Figueiredo Leite. – 2022.

74 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2022.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Busca de Arquiteturas Neurais. 3. Algoritmos Evolucionários de Inspiração Quântica. 4. Classificação de Imagens. I. Bernardes Rebuzzi Vellasco, Marley Maria. II. Tereza Figueiredo Leite, Karla. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. IV. Título.

CDD: 004

To my friends and family
for the support and encouragement.

Acknowledgments

A pandemic Master's is far different from what I expected. It was a huge challenge to move out from my parents house and two months later start the new normal life away from my friends, family and colleagues. Besides learning everything away from the campus and having a physically distant contact with my advisors, I'm glad that I made this far.

With that in mind, first I would like to thank my advisor and my co-advisor Marley and Karla. They made their best to teach and guide me in distance learning. Without them, I wouldn't have so many new opportunities in my life and in my career. Thank you so much for the feedbacks, for the patience and for the encouraging messages.

I also would like to thank my family who are always worried about me and gave me full support and advice from distance as well.

Additionally, I'd like to thank my friends for making me laugh in difficult times and to be by my side for twenty years, always accepting who I am and encouraging me to give my best.

Finally, I would like to thank CNPQ for the financial support. For me the most important thing is to contribute scientifically to society. We are living in times when science is being denied in many ways and it is up to us scientists to continue fighting for its development.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

Abstract

Noce, Julia; Bernardes Rebuzzi Vellasco, Marley Maria (Advisor); Tereza Figueiredo Leite, Karla (Co-Advisor). **Enhanced Q-NAS for Image Classification**. Rio de Janeiro, 2022. 74p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Deep neural networks are powerful and flexible models that have gained the attention of the machine learning community over the last decade. Usually, an expert spends significant time designing the neural architecture, with long trial and error sessions to reach good and relevant results. Because of the manual process, there is a greater interest in Neural Architecture Search (NAS), which is an automated method of architectural search in neural networks. NAS is a subarea of Automated Machine Learning (AutoML) and is an essential step towards automating machine learning methods. It is a technique that aims to automate the construction process of a neural network architecture. This technique is defined by the search space aspects of the architectures, search strategy and performance estimation strategy. Quantum-inspired evolutionary algorithms present promising results regarding faster convergence when compared to other solutions with restricted search space and high computational costs. In this work, we enhance Q-NAS: a quantum-inspired algorithm to search for deep networks by assembling simple substructures. Q-NAS can also evolve some numerical hyperparameters, which is a first step in the direction of complete automation. Our contribution involves experimenting other types of optimizers in the algorithm and make an in-depth study of the Q-NAS parameters. Additionally, we present Q-NAS results, evolved from scratch, on the CIFAR-100 dataset using only 18 GPU/days. We were able to achieve an accuracy of 76.40% which is a competitive result regarding other works in literature. Finally, we also present the enhanced Q-NAS applied to a case study for COVID-19 x Healthy classification on a real chest computed tomography database. In 9 GPU/days we were able to achieve an accuracy of 99.44% using less than 1000 samples for training data. This accuracy overcame benchmark networks such as ResNet, GoogleLeNet and VGG.

Keywords

Neural Architecture Search; Quantum-Inspired Evolutionary Algorithms; Image Classification.

Resumo

Noce, Julia; Bernardes Rebuzzi Vellasco, Marley Maria; Tereza Figueiredo Leite, Karla. **Aprimoração do Algoritmo Q-NAS para Classificação de Imagens**. Rio de Janeiro, 2022. 74p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Redes neurais profundas são modelos poderosos e flexíveis que ganharam a atenção da comunidade de aprendizado de máquina na última década. Normalmente, um especialista gasta um tempo significativo projetando a arquitetura neural, com longas sessões de tentativa e erro para alcançar resultados bons e relevantes. Por causa do processo manual, há um maior interesse em abordagens de busca de arquitetura neural, que é um método que visa automatizar a busca de redes neurais. A busca de arquitetura neural(NAS) é uma subárea das técnicas de aprendizagem de máquina automatizadas (AutoML) e uma etapa essencial para automatizar os métodos de aprendizado de máquina. Esta técnica leva em consideração os aspectos do espaço de busca das arquiteturas, estratégia de busca e estratégia de estimativa de desempenho. Algoritmos evolutivos de inspiração quântica apresentam resultados promissores quanto à convergência mais rápida quando comparados a outras soluções com espaço de busca restrito e alto custo computacional. Neste trabalho, foi aprimorado o Q-NAS: um algoritmo de inspiração quântica para pesquisar redes profundas por meio da montagem de subestruturas simples. O Q-NAS também pode evoluir alguns hiperparâmetros numéricos do treinamento, o que é um primeiro passo na direção da automação completa. Foram apresentados resultados aplicando Q-NAS, evoluído, sem transferência de conhecimento, no conjunto de dados CIFAR-100 usando apenas 18 GPU/dias. Nossa contribuição envolve experimentar outros otimizadores no algoritmo e fazer um estudo aprofundado dos parâmetros do Q-NAS. Nesse trabalho, foi possível atingir uma acurácia de 76,40%. Foi apresentado também o Q-NAS aprimorado aplicado a um estudo de caso para classificação COVID-19 x Saudável em um banco de dados de tomografia computadorizada de tórax real. Em 9 GPU/dias, conseguimos atingir uma precisão de 99,44% usando menos de 1000 amostras para dados de treinamento.

Palavras-chave

Busca de Arquiteturas Neurais; Algoritmos Evolucionários de Inspiração Quântica; Classificação de Imagens.

Table of contents

List of figures

List of tables

List of algorithms

This is my letter to the world
That never wrote to me

Emily Dickinson, *Selected Letters.*

1
Introduction

In the last decade, Deep Learning methods have become popular for solving a variety of tasks, such as image, speech and automatic translation (Hutter, Kotthoff e Vanschoren 2019), (Zoph e Le 2016). In most cases, networks built manually by specialists are responsible for the great success of these applications in the literature (LeCun, Bengio e Hinton 2015). For image application, EfficientNet (Tan e Le 2019), InceptionNet (Szegedy et al. 2016) and VGG(Simonyan e Zisserman 2014) are examples of successful hand-made architectures. However, building a successful network from scratch can be a time-consuming and error-prone process. Because of the cost of the manual process, there is a great interest in Neural Architecture Search, which is an automated method of architectural search in neural networks (Hutter, Kotthoff e Vanschoren 2019).

Automated Machine Learning (AutoML) has become an important research area with wide applications of machine learning techniques. The goal of AutoML is to make the area of machine learning accessible to other scientists who are interested in applying it to different domains. For that, the research areas of AutoML propose to automate decision making when building and training a neural network (Hutter, Kotthoff e Vanschoren 2019).

Neural Architecture Search (NAS) is a subarea of AutoML and is an essential step towards automating machine learning methods. It is a technique that aims to automate the construction processes of a neural network architecture (Hutter, Kotthoff e Vanschoren 2019). Several new algorithms have been proposed to solve the Neural Architecture Search problem, but many of them require significant computational resources. The approach includes different techniques such as Reinforcement Learning (RL) (Zoph e Le 2016), (Baker et al. 2016), (Hsu et al. 2018), (Tian et al. 2020), Bayesian optimization (Han et al. 2020),(Real et al. 2019), evolutionary algorithms (Szwarcman, Civitarese e Vellasco 2019), (Awad, Mallik e Hutter 2020), (Ottelander et al. 2021) or its variation, the quantum-inspired evolutionary algorithms (QIEA), which show promising results when it comes to faster convergence. The first proposal for QIEA used important principles of quantum computing such as the quantum bit, the linear superposition of states and the quantum rotation gate. Empirical results show that QIEAs can find better solutions with fewer evaluations when compared to similar algorithms for many optimization problems (Cruz, Vellasco e Pacheco 2007), (Cruz, Vellasco e Pacheco 2010).

Quantum Inspired Neural Architecture Search (Q-NAS) (Szwarcman, Civitarese e Vellasco 2019) is a QIEA model for searching deep neural architectures, assembling substructures and optimizing numerical hyperparameters. In Q-NAS, quantum individuals contain chromosomes which have two parts: one is responsible for encoding the numerical space of some hyperparameters (such as the weight decay factor) and the other encodes the space for discrete neural architecture. However, this is not a co-evolution context: the algorithm evaluates classical individuals as a solution containing the architecture and the hyperparameters. The best accuracies found on the CIFAR-10 task were 93.85% for a residual network and 93.70% for a convolutional network, overcoming hand-designed models and some NAS works. Considering the addition of a simple early-stopping mechanism, the evolution times for these runs were 67 GPU days and 48 GPU days, respectively. Also, they applied Q-NAS evolved network to CIFAR-100 without any parameter adjustment, reaching an accuracy of 74.23%, which is comparable to a ResNet with 164 layers.

1.1 Objectives

Our primary goal is to investigate and evaluate enhancements in Q-NAS in order to improve performance in terms of accuracy and processing time. For this, we evaluate the algorithm by training the network on CIFAR-10 and CIFAR-100 benchmark datasets as a way to mediate and compare with the originally proposed Q-NAS. Moreover, we evaluate the previous Q-NAS version and its evolution strategy and compare to the proposed Neural Architecture Search strategies. Our focus is to analyze the impact of changing the parameters in CIFAR-10 and CIFAR-100 datasets. As described in following chapters, we observe that changing only the optimizer in the Q-NAS training phase will bring positive impacts on the accuracy.

As mentioned in Section 1, the work of Szwarcman, Chevitarese, and Vellasco (Szwarcman, Civitarese e Vellasco 2019) achieved an accuracy of 74.23% for the CIFAR-100 database. Thus, our second goal is to train a network from scratch so that it can achieve similar or even surpass the accuracy found in the previous proposal in a much more restricted computational infrastructure.

The COVID-19 pandemic changed our way of living. With its rapid spread and potential virulence, many lives are compromised and hospitals are dealing with capacity management to treat as many patients as possible. An efficient way to detect the presence of the virus and the patient's condition is the chest computed tomography images which is analyzed by radiologists to look for visual indicators associated with a viral infection. Because of this scenario, the final goal of this work is also to apply the algorithm to a real database of chest computed tomography images to build an architecture which detects Sars-CoV-2 of patients from Pedro Ernesto University Hospital.

1.2 Contributions

The main contributions of this work involve carrying out a study of the best individual found at (Szwarcman, Civitarese e Vellasco 2019) and making an assessment of possible improvements for this individual. For this case, we investigated and applied other optimizers and with that we were able to achieve an even greater accuracy than the previous work. One should note that this improvements were made in the retraining phase, which means that we already picked the best individual found by the previous work and improved it in the retraining phase.

Furthermore, it involves providing an in-depth study of the Q-NAS for the CIFAR-100 database so that it is possible to search an architecture that surpasses the result found in (Szwarcman, Civitarese e Vellasco 2019).

Finally, as far as we are concerned this is the first quantum-inspired NAS work that is applied to the task of detecting COVID-19 through computed tomography image classification. We were able to achieve an accuracy of 99.44% in the classification task.

1.3 Work Outline

This work comprises five additional chapters, which we describe below. In Chapter 2, we provide the theoretical background necessary to understand the NAS context, including the concepts of convolutional neural networks and a review of some NAS works.

We present quantum-inspired evolutionary algorithms in Chapter 3, including the definition of essential concepts and a brief description of some previous works.

In the next chapter 4, we revisit Q-NAS proposed by (Szwarcman 2020) and provide an in-depth study of the previous experiments in order to outperform them. We investigate the previous experiments to make a comparison and to analyze the main contributions that changing the Q-NAS parameters could possibly have in the accuracy.

In chapter 5, we point out the enhancements made in Q-NAS algorithm, emphasizing the main changes and make an in-depth study in the parameters.

Next, Chapter 6 describes and discusses the experiments. We present our experimental track in the order it was developed, with each section addressing a specific investigation subject.

Finally, we present the final remarks in Chapter 7, along with the next steps of our research.

2
Neural Architecture Search

As previously reported, there are different neural architecture search techniques for image classification. The techniques covered in this section will be: Reinforcement Learning (RL), Evolutionary Algorithms and NAS with different methods.

2.1
NAS with Reinforcement Learning

Reinforcement Learning approaches are useful for modeling a sequential decision-making process in which an agent interacts with an environment with the main objective of maximizing its future rewards (Sutton e Barto 2018). The agent learns to improve its behavior according to multiple interactions with the environment of the problem (Wistuba, Rawat e Pedapati 2019). A policy defines the behavior of the learning agent at any given time: it maps the perceived states of the environment to action decisions when in those states. The NAS is formulated as a Reinforcement Learning problem by considering the generation of a neural architecture as the agent's action. Moreover, the environment of the problem is the same as the network and hyperparameters search problem. The RL approaches are mainly distinguished by the agent's policy representation and the method to optimize it.

In (Baker et al. 2016), the authors introduce MetaQNN, which uses Q-Learning, reinforcement technique with epsilon-greedy as exploration strategy, (Watkins 1989) to train a policy that selects CNN layers sequentially. For example, if the layer is convolutional, the options for kernel size and filters are $\{1, 3, 5\}$, and $\{64, 128, 256, 512\}$. In addition, Zhong et al. (Zhong et al. 2018) present one of the first approaches implementing block-wise architecture search, the BlockQNN, which automatically builds convolutional networks using Q-Learning (Mnih et al. 2015). The block structure is similar to ResNet and Inception (GoogLeNet) networks since it contains shortcut connections and multi-branch layer combinations.

Additionaly, in (Chu, Zhang e Xu 2020), the authors proposed MoreMNAS (Multi-Objective Reinforced Evolution in Mobile Neural Architecture Search). They incorporate a variant of multi-objective genetic algorithm, in which the search space is composed of various cells and each cell contains an amount of repeated basic operators. For example, two repeated 3×3 2D convolutions with 16 channels is a basic operator. Thus, the crossovers and mutations can be performed at the cell level. Moreover, reinforced control is mixed with a natural mutating process to regulate arbitrary mutation, maintaining a delicate balance between exploration and exploitation.

2.2 NAS with Evolutionary Algorithms

Evolutionary algorithms (EA), as the name suggests, are inspired by the natural process of evolution. A population of individuals represents candidate solutions to an optimization problem. All the individuals are evaluated to assign a fitness value to them. In each iteration of the algorithm (generation), the fittest individuals are selected to create a new population (Eiben, Smith et al. 2003). One common method of selecting parents who will generate new individuals in Neural Architecture Search is tournament selection (Goldberg e Deb 1991). The tournament selection made k tournaments in order to select the individuals to crossover. Each tournament is made with n individuals (usually 2 individuals) randomly selected and the one with best fitness is chosen for the recombination stage. (Real et al. 2019) and (Real et al. 2017) applied tournament selection in their works. In (Real et al. 2017) the authors propose an evolutionary method that produces a fully trained network without the need for retraining. The authors start the evolution process from a single layer network and apply mutation operators that act on the structure, allowing its growth. In addition, in (Real et al. 2019) the authors apply an evolutionary algorithm to search for normal and reduction cells in the NAS-Net search space, as defined in (Zoph et al. 2018). New cells are generated by parent individual cells that have been mutated, with simple modifications. Its mutation operator ensures that the child cells are still in NAS-Net space. Instead of removing the worst individuals, they eliminate the oldest ones in each iteration.

Furthermore, EAs for multi-objective evolutionary NAS (ENAS) are gaining more and more attention from researchers. The single objective ENAS algorithms are always concerned about only one objective, e.g., the classification accuracy, and these algorithms have only one goal: searching for the architecture with the highest accuracy. In general, most of the multi-objective ENAS algorithms aim at dealing with both the performance of the neural network and the number of parameters simultaneously (Lu et al. 2019), (Elsken, Metzen e Hutter 2018), (Yang et al. 2020). However, these objective functions are often in conflict with each other. For example, getting a higher accuracy often requires a more complicated architecture with the need of more computational resources. On the contrary, a device with limited computational resource, e.g., a mobile phone, cannot afford such sophisticated architectures (Liu et al. 2021).

The simplest way to tackle the multi-objective optimization problem is by converting it into a single objective optimization problem with weighting factors, i.e., the weighted summation method. The Equation 2-1 is the classical linear form to weight two objective functions f_1 and f_2 into a single objective function, where the $\lambda \in (0, 1)$ denotes the weighting factor.

$$F = \lambda f_1 + (1 - \lambda) f_2 \tag{2-1}$$

In (Zhang 2019), (Laredo et al. 2019), (Loni et al. 2018), the multi-objective optimization problem was solved by using the available single objective optimization methods by Equation 2-1 of the weighted summation. This works consist in finding an architecture for image classification task. Chen et al.(Chen et al. 2018) did not adopt the linear addition as the objective function, whereas using a nonlinear penalty term. However, the weights manually defined may incur bias (Deb 2014).

A Different type of the Evolutionary NAS method is proposed by (Li e Talwalkar 2020). Their algorithm is designed for an arbitrary search space with a Directed Acyclic Graph (DAG) representation. In order to combine random search with weight-sharing, they simply use randomly sampled architectures to train the shared weights. In the CIFAR-10 database, they achieved an accuracy of 97.29%.

Finally, in recent work (Ottelander et al. 2021), the authors use a local search strategy. Their work considers multi-objective NAS (the objective is a function of accuracy and network complexity), focuses on macro search rather than cell-based search.

2.3 NAS with Other Methods

Compared with the above gradient-free optimization methods, the gradient-based (GD-based) methods have become increasingly popular recently, mainly because their search speed is much faster than RL-based and EA-based methods (Zhu, Zhang e Jin 2021). Early GD-based methods (Shin, Packer e Song 2018), (Ahmed e Torresani 2018), (Fang et al. 2020) implement this idea for optimizing layer hyperparameters or connectivity patterns, respectively. Lorraine et al. (Lorraine, Vicol e Duvenaud 2020) introduce an algorithm for inexpensive GD-based hyperparameter optimization. Liu et al. (Liu, Simonyan e Yang 2018) employ GD in the DARTS algorithm, optimizing both the network weights and the architecture. The authors use relaxation tricks to make a weighted sum of candidate operations differentiable, and then apply the gradient descent method to train the weights directly. Inspired by DARTS (Liu, Simonyan e Yang 2018), Dong et al. (Dong e Yang 2019) introduce gradient-based search using the differentiable architecture sampler (GDAS) method. The authors develop a method that samples individual architectures in a differentiable way to accelerate the architecture search procedure.

One bottleneck of the above GD-based NAS methods (e.g. DARTs) is that it requires excessive GPU memory during the search in that all candidate network layers must be explicitly instantiated in the GPU memory. As a result, the size of the search space is constrained. To address this issue, Wan et al. (Wan et al. 2020) propose DMaskingNAS, a memory and computationally efficient DARTS variant. DMaskingNAS employs a masking mechanism for feature map reuse (Zhu, Zhang e Jin 2021).

Another way to address the above problem is to utilize proxy tasks, e.g., learning with only a small number of building blocks or training for a small number of epochs (Liu, Simonyan e Yang 2018) (Xie et al. 2018). However, these approaches cannot guarantee to be optimal target tasks due to the restricted block diversity (Cai, Zhu e Han 2018). Cai et al. (Cai, Zhu e Han 2018) proposed the ProxylessNAS method, which directly designs the networks based on the target task and hardware instead of proxy. Meanwhile, the authors used path binarization to reduce the computational cost (GPU-hours and GPU memory) of NAS to the same normal training level. Hence, the ProxylessNAS algorithm can generate network architectures on the ImageNet dataset without any proxy (Zhu, Zhang e Jin 2021).

A more recent NAS study is called Federated Neural Architecture Search, which aims to optimize the architecture of neural network models in the federated learning environment. The main purpose of federated learning is to protect users' private information, while distributed learning aims to accelerate training speed. Second, federated learning cannot determine the data distribution of any client devices. Moreover, federated learning is often categorized based on the distribution characteristics of the data.

Most NAS methods include two steps: i) searching the architecture of the neural network model; and ii) training the weights of the found neural network model afterwards. Additionally, and most importantly, only the final performance is considered for comparison with other methods.. (Zhu, Zhang e Jin 2021).

One of the federated NAS method is presented as adversarial federated neural architecture search, and it has two purposes: (1) inference of the client data information; (2) attack the global model to conduct backdoor

(Bagdasaryan et al. 2020) elements or even let the model unusable. Geiping et al. (Geiping et al. 2020) showed that local images can be reconstructed from the knowledge of model parameters (or gradients) by inverting gradients techniques. In addition, an adversarial GAN (Goodfellow et al. 2014) generator can be developed on either the server (Wang et al. 2019) or the client side (Hitaj, Ateniese e Perez-Cruz 2017). The adversary can reconstruct other participating clients' private data even if it has no knowledge of the label information. In general, finding robust model architectures in federated learning to defend against adversarial attacks is still a hard task.

2.4 NAS applied in COVID-19 Scenario

One of the contributions of this work is to provide an algorithm capable to detect COVID-19 in chest computed tomography (CT) images. Because of this, we are providing all related works, as far as we are concerned, that apply NAS to detect COVID-19.

(He et al. 2021) proposes an efficient Evolutionary Multi-objective neural ARchitecture Search (EMARS) framework, that automatically searches for 3D neural architectures based on a search space for COVID-19 chest CT scan classification. Within the framework, they proposed a factorized 3D search space, in which all child architectures use weight sharing among each other to significantly improve the search efficiency and finish the search process in 8 hours. In their search space, a global average pooling layer is inserted before the fully connected layer; therefore, the class activation mapping (CAM) algorithm, which is a method to improve explainability, can be easily embedded into their searched models, which can help doctors locate the discriminate lesion areas on the CT scan images. In this paper, they use three publicly available datasets: Clean-CC-CCII (He et al. 2020), MosMedData (Morozov et al. 2020) and COVID-CTset (Rahimzadeh, Attar e Sakhaei 2021), all of which provide 3D chest CT scans. They reached the accuracy of 89.61% in 1.3 GPU days.

Additionally, in (Timofeev, Chrysos e Cevher 2021) they propose a NAS-based framework that bears the threefold contributions: (a) they focus on the self-supervised scenario, i.e., where no labels are required to determine the architecture,(b) they assume the datasets are imbalanced, (c) they design each component to be able to run on a resource constrained setup, i.e., on a single GPU. The authors also conduct experiments on ChestMNIST (Wang et al. 2017), which contains 78,468 images of chest X-ray scans and COVID-19 X-ray (Ozturk et al. 2020).

Finally in (Rahbar e Yazdani 2021), they first introduce a new dataset with augmented features and then forecast COVID-19 cases with a new approach, using an evolutionary neural architecture search with Binary Bat Algorithm (BBA) (Yang 2010) to generate an optimized deep recurrent network to forecast the pandemic cases. The individuals are defined using a hybrid encoding structure as the population of BBA. Since the final goal is forecasting COVID-19 daily cases, one can conclude that the problem is regression. As the literature of artificial neural networks and deep learning models in regression problems suggests, the authors select Mean Squared Error (MSE) as BBA's fitness function. As a result, the authors built a network that had Mean RMSE Loss of 1.23e-3, using only one GPU.

2.4.1 Final Remarks

Most of the works reported above use more than 50 GPUs/day to do this search. One of them is NASNet, the authors trained in the dataset CIFAR-10 using 500 GPUs which makes its use very limited in terms of scalability. Moreover, in (Baker et al. 2016), the authors searched for an architecture for CIFAR-10 that used 10 GPUs for 10 days (100 GPUs/days). In (Ottelander et al. 2021), the authors were able to find a network that uses only 1 GPU in 6.5 minutes. However, their search space is much more limited. The reduction layers such as *Max-pooling* are fixed in the architecture and there are only 5 types of convolutional layers. In addition, the authors compare their Local Search with the Random Search approach. They do not provide further information along others NAS strategies.

In the QIEA context, the authors (Ye et al. 2020) also encode a CNN using quantum inspiration. However, they fixed the first layer to be a Convolutional Layer. Moreover they also fixed the length size of the convolutional layers by 10. This means that the authors also limited the search space in terms of complexity. Finally, the authors do not penalize the reducing layers, which makes the evolution capable to find irregular structures.

The Q-NAS comparative results with other works are described in Table 2.1 for CIFAR-10 while 2.2 compares Q-NAS result with other works using CIFAR-100 dataset.

Hand-designed models	accuracy(%)	#params	GPU days
ResNet (He et al. 2016)	93.57	1.7M	-
VGG (Simonyan e Zisserman 2014)	92.06	15.2M	-
GoogleLeNet (Szegedy et al. 2015)	93.64	-	-
NAS			
Meta-QNN (Baker et al. 2016)	93.08	11.18M	100
DARTS (Liu, Simonyan e Yang 2018)	97.24	3.3M	5
NAS-Net (Zoph et al. 2018)	96.86	3.3M	2000
Block-QNN-S (Zhong et al. 2018)	96.46*	39.8M	96
Q-NAS (Szwarcman 2020)	93.85	7.07M	67

Table 2.1: Comparing our results with some literature models. The ‘*’ marks the methods that used other datasets for the search and applied the network on CIFAR-10.

Hand-designed models	accuracy(%)	#params	GPU days
ResNet-1001 (He et al. 2016)	77.30	10.2M	-
ResNet-164 (He et al. 2016)	75.67	1.7M	-
Network in Network (NiN) (Lin, Chen e Yan 2013)	64.32	-	-
NAS			
Meta-QNN (Baker et al. 2016)	72.86*	11.18M	100
Block-QNN-S (Zhong et al. 2018)	81.94	39.8M	96
Q-NAS (Szwarcman 2020)	74.23%	6.25M	156

Table 2.2: Results from the literature on CIFAR-100. The ‘*’ marks the methods that used other datasets for the search and applied the network on CIFAR-100.

3
Quantum-inspired Evolutionary Algorithms

3.1
Basics of Quantum Computing

In conventional computations, information is recorded at a macroscopic level and represented using two logical basis states; a "0" or "1". These states form a bit of information coded in the current flowing in a circuit: closed for "0" and open for "1". However, quantum computations are represented at a microscopic level using quantum states. Inspired by quantum mechanics, this allows information to be represented using Dirac notation: in the ground state as $|0\rangle$ and in the excited state as $|1\rangle$ (Schumacher 1995). These are the two basis states that form a q-bit.

In quantum computer, the q-bits are quaternary in nature. Each q-bit has three states unlike the binary bits in classical computers. The states are state 0 or state 1 and the linear superposition of these two basic states. The state $|\Psi\rangle$ of the q-bit $\begin{bmatrix} \alpha & \beta \end{bmatrix}^T$ is defined as (Han e Kim 2002):

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α and β are complex numbers and represent the probability amplitudes of q-bit to be 0 and 1, respectively. The q-bit collapses to the state "0" with probability α^2 and to the state "1" with probability β^2. The q-bit state can be modified by means of quantum gates.

A quantum gate is a unitary operator that acts on the q-bit basis. It can be represented by a matrix M that, in order to preserve orthogonality, must satisfy $M^\dagger M = I$, where $M^\dagger$ is the conjugate transpose of M (Han e Kim 2002). One can think of these unitary transformations as being rotations of the q-bit vector space. This is the concept of a q-bit giving a single classical bit of information, which collapses from the quantum state to one of the two classical states $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ or $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The difference between the memory in classical and quantum computation is that n bits in a classical computation require 2^n binary states to represent them; however, the n q-bits require only n quantum states.

3.2 Quantum-Inspired Evolutionary Algorithms

The idea of quantum-inspired computing is to create classical algorithms, which can be executed in classical computers, but that take advantage of the paradigms of quantum physics (Moore e Narayanan 1995), (Cruz 2007). A quantum-inspired evolutionary algorithm (QIEA) applies quantum computing principles, such as quantum bits and superposition of states to solve optimization problems. The authors in (Han e Kim 2002) presented the first proposal of quantum-inspired evolutionary algorithms where some important principles of quantum computing are used, such as the collapsed quantum bit, the linear superposition of states and the quantum rotation gate.

A QIEA, just like other evolutionary algorithms, is defined by the individual representation, an evaluation function, and the population dynamics. However, in QIEA, the individual is encoded in a probabilistic fashion, thus representing a superposition of states (solutions) in the search space (Han e Kim 2002).

The first practical QIEA used a q-bit representation, in which a string of q-bits defines an individual. Initially, a q-bit individual represents all possible states with the same probability, thus better characterizing the population diversity than other types of representations. During evolution, a Q-gate operator can modify the probability of each q-bit, so it gradually converges to a single state – the optimal solution. However, quantum individuals cannot be directly evaluated: they must be observed to generate classical individuals. In other words, since a quantum individual represents many quantum states, it can only be evaluated when it collapses to a single one (Han e Kim 2002) (Szwarcman 2020).

One key concept of QIEAs that distinguishes them from other EAs is the quantum population. It represents a superposition of states covering the search space, or more specifically, each quantum state characterizes a possible solution. The Algorithm 1 shows a QIEA pseudocode (Han e Kim 2002). The QIEA procedure includes some unique steps, such as the observation of the quantum population Q(t) and the update of the quantum individuals.

QIEAs combine binary and real representation (Pinho, Vellasco e Cruz 2009) and also have been proposed for problems in which both real and categorical parameters need to be evolved, such as the hyperparameters and weights of a neural network (Cardoso et al. 2015). The numerical variables are represented using the square pulse scheme, and the categorical parameters follow the q-bit scheme. The variation operators (Q-gate, crossover or mutation), that will be described in the following Sections, are applied accordingly to the problem. As

Algorithm 1: QIEA pseudocode

```
$t \leftarrow 0$
Initialize $Q(t)$
Generate classical population $P(t)$ observing $Q(t)$
Evaluate $P(t)$
Store the best solutions of $P(t)$ in $B(t)$
while $t \leq T$ do
  $t \leftarrow t + 1$
  Generate classical population $P(t)$ observing $Q(t-1)$
  Evaluate $P(t)$
  $Q(t) \leftarrow$ Update $Q(t)$
  Store the best solutions of $P(t)$ and $B(t-1)$ in $B(t)$
end while
```

described, one q-bit can be represented as $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$. And m-q-bits representation is defined as:

$$q_i^t = \left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & \cdots & \beta_m \end{array} \right] \tag{3-1}$$

where $|\alpha_i|^2 + |\beta_i|^2 = 1, i = 1, 2, \cdots, m$

This representation have an advantage that is the possibility to represent every superposition states. In (Han e Kim 2002) they define a quantum population with N individuals $Q(t) = \{\boldsymbol{q}_1^t, \boldsymbol{q}_2^t, \ldots \boldsymbol{q}_N^t\}$ Each quantum individual q_i^t is a string of q-bits. The quantum individual q_i^t can represent a solution of 2^m states.

Suppose that we have a quantum individual consisting of two q-bits:

$$\left[\begin{array}{c|c} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{array} \right] \tag{3-2}$$

To calculate the probability of the state $|01\rangle$, we should first calculate the probability amplitude associated with this state. The amplitude is obtained by multiplying the positions $(0, 0)$ and $(1, 1)$ of the quantum individual matrix. So the probability of the state $|01\rangle$ is given by:

$$P_{|01\rangle} = \left(\frac{1}{\sqrt{2}} \times \frac{-1}{\sqrt{2}} \right)^2 = \frac{1}{4} \tag{3-3}$$

In this example, one can see that a single quantum individual can represent four possible solutions. The classical binary representation would need four strings to represent the same amount of information.

Now that the representation part has been briefly explained, we will proceed with the algorithm step-by-step, following the pseudocode presented in the Algorithm 1. At first, we initialize the quantum population $Q(t)$ (line 2 in the Algorithm 1). The strategy is to assign initial values to each quantum gene j of every individual i, so that all possible states have the same initial probability. This can be described below:

$$\alpha_{ij}^{0}, \beta_{ij}^{0} = \frac{1}{\sqrt{2}} \quad i = 1, 2, \dots, N; j = 1, 2, \dots G \tag{3-4}$$

Next, we generate classical individuals by observing the quantum individuals. The observation is an analogy as we are working with classical computers. Consequently, to generate a binary solution x_j^t to be evaluated, one selects a state for each gene independently.

Thus, we want to generate P(0) observing Q(0) (line 3 in Algorithm 1), where $P(0) = \{\mathbf{x}_1^0, \mathbf{x}_2^0, \dots, \mathbf{x}_n^0\}$ at generation $t = 0$. One binary solution, $\mathbf{x}_j^0, j = 1, 2, \dots, n$ is a binary string of length n, which is formed by selecting either 0 or 1 for each bit, respecting the probabilities $|\alpha_i^0|^2$ and $|\beta_i^0|^2$ (Han e Kim 2002). One should point out that a quantum computer collapses to a single state in observing a quantum state. Finally, this process is repeated for all N individuals to generate the classical population.

Once generated, the first classical population P(0) is ready for evaluation to give a level of its fitness. The best solutions of one generation are stored into B(0), where $B(0) = \{\mathbf{b}_1^0, \mathbf{b}_2^0, \dots, \mathbf{b}_n^0\}$. At generation 0, B(t) = P(t). This ends the initial procedures, and we begin the loop of generations by the observation process followed by the evaluation step.

After, the q-bit individuals in $Q(t)$ are updated by applying Q-gates proposed by (Han e Kim 2002) and it is described below:

$$U(\Delta\theta_i) = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \tag{3-5}$$

where $\Delta\theta_j$ is a rotation angle for each q-bit toward either 0 or 1 state depending on its sign. $\Delta\theta_j$ should be designed in compliance with the application problem.

A Q-gate is defined as a variation operator of QIEA, which the updated q-bit should satisfy the normalization condition, $|\alpha_i'|^2 + |\beta_i'|^2 = 1$, where α' and β' are the values of the updated q-bit. One should note that the best solutions influence the quantum individuals' update, so the quantum gate will gradually rotate the q-bits toward promising solutions. Consequently, the q-bits will progressively converge to a single state which means the possible optimal solution, as we can't guarantee that it achieves the optimal solution. This means that QIEA starts with a global search and changes automatically into a local search which leads to a good balance between exploration and exploitation (Han e Kim 2002).

Finally, the last step in the loop consists of storing the best solutions among $B(t-1)$ and $P(t)$ into $B(t)$. Then, the solutions from the current generation and the best solution are ordered by fitness value and the best ones from the group are selected.

QIEA was implemented by other researches with different versions to solve several other problems. In (Ye et al. 2020), the authors encode CNNs into quantum chromosomes, then, the quantum chromosomes are updated by applying quantum gates and finding the best individual with a quantum genetic algorithm. Finally, the algorithm predicts the network performance after a few stochastic gradient descent steps by means of an evaluation estimate strategy, in order to avoiding overfitting and to speed up the evolution process.

Additionally, in (Saad et al. 2021), the authors investigate the performance of a Quantum-Inspired Genetic Algorithm(QIGA) that was adapted to solve the Resources-Constrained Project-Scheduling Problem (Blazewicz, Lenstra e Kan 1983). They analysed the effects of different quantum parameters in the QIGA, such as quantum population size, quantum mutation rate, and single-point and two-point quantum crossover, initialised the quantum population using quantum rotation gates with different rotation angles. Their result was competitive when compared to solutions using heuristics and meta-heuristics.

Moreover, the authors (Mittal et al. 2020) proposed a feature selection method based on QIEA for the classification of fake-face images. In the proposed Improved QIEA (IQIEA), the authors add a mutation operation in the rotational quantum-gate to enhance the exploration ability. The proposed method initially extracts features of an image through AlexNet, a pre-trained deep learning model. The extracted features are processed through the proposed improved quantum-inspired evolutionary algorithm to select an optimal feature subset. Finally, the classification is performed with the elicited feature subset through the kNN classifier.

Finally,(Ramos e Vellasco 2020) applied q-bit QIEA for feature selection in a classification task of electroencephalography data. They compared the QIEA approach with a classical genetic algorithm. The results for the implemented QIEA demonstrated a greater convergence speed. The authors conclude that it can also be an effective algorithm for feature selection.

4
Quantum-Inspired Neural Architecture Search

This chapter explains the Quantum-inspired Neural Architecture Search (Q-NAS): an algorithm proposed by (Szwarcman 2020). Q-NAS is a quantum-inspired evolutionary algorithm that automates the process of building a deep neural network to execute a predefined task.

In Q-NAS, quantum individuals contain chromosomes that have two parts: one is responsible for encoding the numerical space of some hyperparameters (such as the weight decay factor) and the other for encoding the discrete architecture space. This approach goes in the direction of complete automation, which is the goal of AutoML. Moreover, one quantum individual can generate multiples classical individuals and then, the algorithm evaluates these classical individuals as a solution containing the architecture and the hyperparameters.

However, as stated by the authors in (Szwarcman 2020), the hyperparameter evolution does not provide better results on CIFAR-10 and CIFAR-100 datasets when compared with the Tensorflow's default hyperparameters values (Abadi et al. 2015) on the same architecture. Thus, this work will not focus on hyperparameters' evolution.

The search space for the network structure is defined by the user and the values of the hyperparameters are fixed for every network structure found by the algorithm. The network search space consists of the selection of building blocks (layer functions) that will be used to assemble the networks. The system's output, is a network description along with the fixed hyperparameters' values that the user defined. This is described in Figure 4.1. The Fx1 ... FxL represent the network function that are in the search space. Q-NAS find the best combination between this functions in order to reach the highest accuracy in the classification task.

One should notice that Q-NAS does not evolve the weights of the networks. Regular gradient based training is conducted for this purpose, as will be described in the following sections. Moreover, as described before, we will only focus on the encoding of the network structure and the values of hyperparameters are fixed.

Figure 4.2 shows a simplified Q-NAS flowchart (Szwarcman 2020): we iterate over the generations until the maximum value defined by the user. Inside the loop, we sample solutions from the quantum individuals, evaluate them, and update the quantum population. The gray boxes indicate the steps that

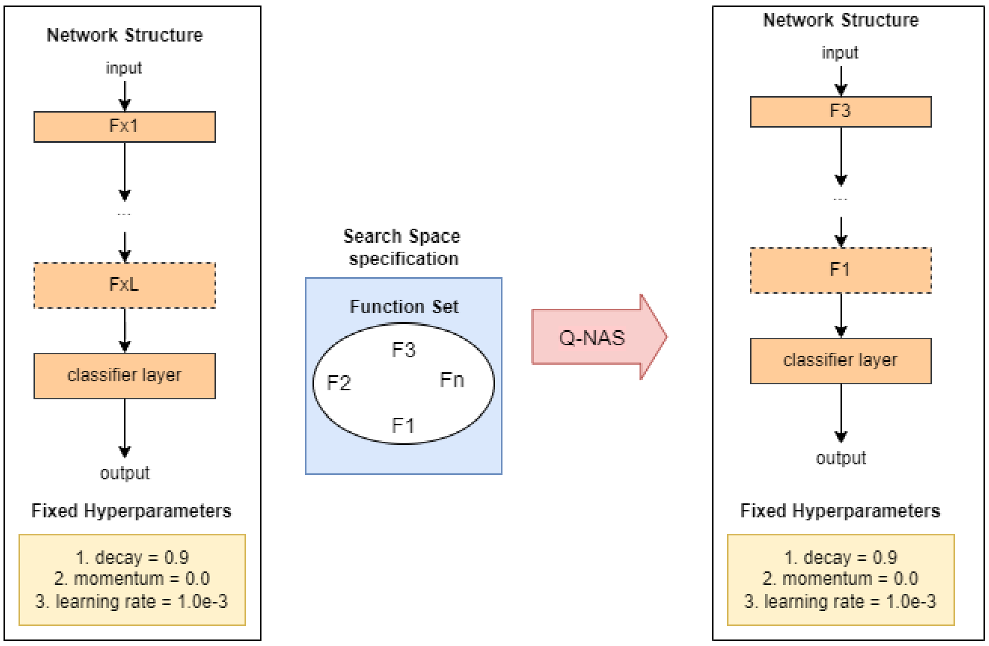

Figure 4.1: Q-NAS context in a classification tasks (Szwarcman 2020)

must consider the particular quantum representation. The quantum individual representation is the main factor that discriminates it from the previously discussed QIEA. The section 4.1 discuss the network structure representation in a quantum individual.

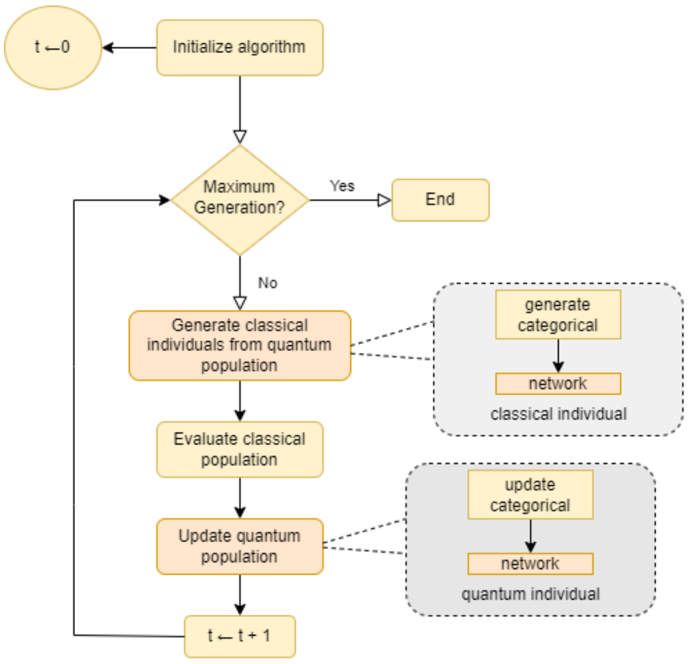

Figure 4.2: Q-NAS flowchart (Szwarcman 2020)

4.1
Q-NAS Network Representation

In (Szwarcman 2020), the authors proposed to represent the network structure in a *chain-like* structure with a fixed size L, in which every node has a function associated with it. These functions are basically designed to be one or more network layers function. For example, one node can be one Convolutional layer in the structure. As the task of this work is image classification, the last network layer is fixed to be a classifier (fully connected) layer. Figure 4.3 shows the *chain-like* structure and some examples of functions. The schematic of the network on the left of the Figure 4.3 indicates the structure of the network, having as the first layer the one indicated with the label F1, until the last one indicated with the label FL, where L is a value previously defined by the user in the algorithm input. Thus, the number of layers is fixed. The last layer, fully connected, is standard in any architecture evaluated, in order to address the problems of pattern classification. At the end (output) the value of the output class is presented. In addition, on the right of Figure 4.3 an example of architecture explored in the search process can be observed, containing 3 layers that represent a sequence of functions: F1, F2 and F3.

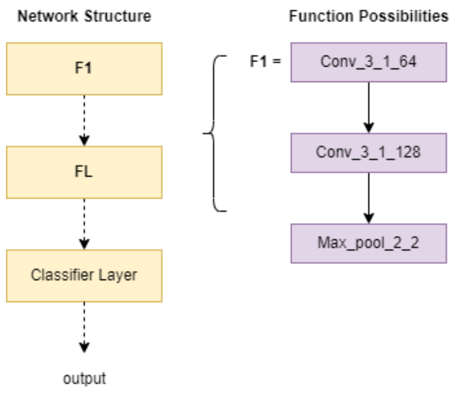

Figure 4.3: Network representation and function possibilities. The functions for each node can be as simple as one unique layer or a more complex structure (Szwarcman 2020).

The only restriction specified by the authors is that a node only can be connected in a chain-like structure; no skip-connections between nodes are allowed. The user can manually specify a list of predefined functions with different kernels, filters and strides that can be selected by the algorithm and be in the search space for every node. To represent variable length networks, the authors included a "no operation" function(NoOp) to the node in the function list.

The predefined function names are mapped to integers in the range $[0, M-1]$. We can define our classical individual p_i as an array of integers:

$$\boldsymbol{p}_i = [g_{i1}, \ldots, g_{iL}]\,; \quad g_{ij} \in [0, M-1] \tag{4-1}$$

where L is the number of nodes in our networks, and M is the number of functions available in the search space. It is important to note that the user can previously define the number of L and M which means that if L and M increase, the computational time may also increase because of the increasing of the complexity of the structure. However, it is also important to note that if we have few complex functions in a structure, it can cost more computational time than have many simpler functions. The quantum gene defines a probability mass function (PMF) for a node in the structure. A quantum gene encoding a single node is then represented by an array described below:

$$\boldsymbol{g}_j = [x_{j1}, \ldots, x_{jM}]\,; \quad x_{jk} \in [0.0, 1.0); \quad \sum_{k=1}^{M} x_{jk} = 1.0 \tag{4-2}$$

If there are N quantum individuals, a maximum network size of L layers and M functions in the function list, the quantum population will be an array of shape *(N, L, M)*. If the user does not specify the initial probability value manually in each function, all nodes will start with the same PMF (Szwarcman 2020). This means that it is possible to start the evolution giving an initial bias to one or more functions.

Figure 4.4 summarizes the entire generation process for the network part of the chromosome: from the user input parameters to the final decoded network. The Q-NAS' parameters are listed on the left. *Conv(k; s, f)* stands for a convolution layer with kernel size *k x k*, stride *s* and *f* filters. The observation of the quantum individual is carried out by sampling from the PMF of each node. The decoding process is a mapping from integers to function names. The final architecture includes a fully connected (FC) classifier layer at the end of the structure.

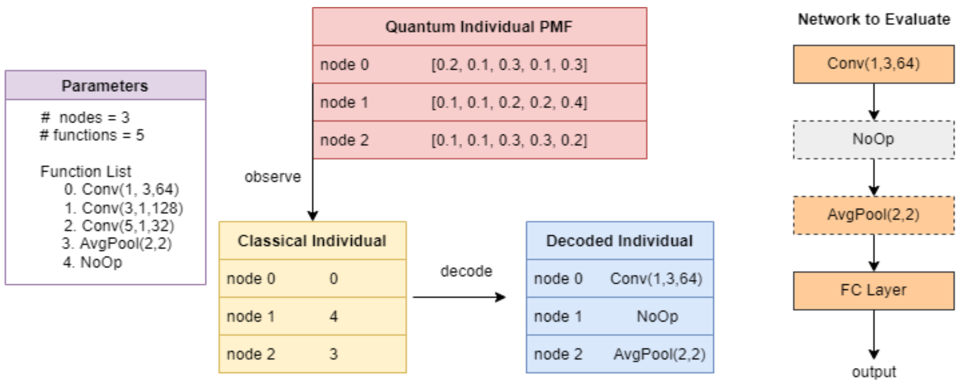

Figure 4.4: Q-NAS Network Quantum Individual Representation Scheme(Szwarcman 2020).

Network quantum individuals can lead to invalid structures. One can assume that there is a pooling operation in the function list that reduces the feature map size in half. For a given input image size, there is a maximum number of times this pooling function can appear in the network before the feature map reaches unit pixel size. However, since Q-NAS samples each node independently, this cannot be handled directly.

Considering these points, the authors (Szwarcman 2020) decided to penalize invalid architectures. First, the authors developed a simple procedure to correct an invalid structure: when building the decoded network for evaluation, they ignore all pooling operations that appear after the allowed number is reached but penalizes the fitness value (Szwarcman 2020). The parameter *penalize_number* defines the maximum number of reducing layers a network can have without being subject to penalization. This means that every pooling layer added, beyond the limit, in the network structure will decrease the fitness value by 0.01, considering that the fitness range is $[0, 1]$. Suppose that we have the parameter *penalized_number* $= 2$ and the network has 5 reducing layers. Its fitness will be reduced by 0.03. This means that instead of waiting for correctly sampled individuals, this approach corrects them at evaluation time, so they can be trained and then penalized. Moreover, it differentiates networks with a few invalid layers from others that greatly exceed the *penalize_number*.

The quantum update procedure must be created for the network part of the chromosome. In Q-NAS, the best classical individuals information modify the quantum population.

For the Q-NAS network quantum update, the heuristic is described below. The loop is only depicted here for clarity; the actual program applies the operation in the entire array of individuals at once. The goal is to increase the probability of a promising function in a node by a random factor, which can assume the maximum value of 0.05. The other probabilities in the same node must be reduced to guarantee the total sum of 1.0. Consequently, the other probabilities are decreased proportionally to their current size. This proportional decrease ensures that small probabilities will never get negative. The maximum update value was chosen to be conservative.

4.2 Q-NAS Algorithm

The Q-NAS algorithm comprises three main operations: (1) population generation; (2) candidate evaluation; (3) ranking and update. In order to find the best individual, the algorithm repeats these steps for T generations (Szwarcman, Civitarese e Vellasco 2019). The Algorithm 3 provides the summarized steps of the Q-NAS. Here, we will refer to the quantum population as

Algorithm 2: Q-NAS network quantum update

```
Generate random mask, based on $update\_quantum\_rate$
Chosen nodes positions = $idx$
for each node position $i$ in $idx$ do
    Get $best\_classical\_individuals[i]$ function $f$
    Increase the probability for $f$ in node $i$ by:
        $update\_value = random() * 0.05$
    Subtract $update\_value$ from the probabilities other than $f$ in node $i$
    proportionally to their current size
end for
```

Q(t), which consists of a set of N quantum individuals $q_i^t, i = \{1, 2, \ldots, N\}$.

Algorithm 3: Q-NAS algorithm

```
$t \leftarrow 0$
Initialize $Q(t)$
while $t \leq T$ do
    Generate classical population $C(t)$ observing $Q(t)$
    if $t = 0$ then
        Evaluate $C(t)$
        $P(t) \leftarrow C(t)$
    else
        $C(t) \leftarrow$ recombination between $C(t)$ and $P(t)$
        Evaluate $C(t)$
        $P(t) \leftarrow$ select$(C(t), P(t))$
    end if
    $Q(t+1) \leftarrow$ update $Q(t)$ based on $P(t)$ values
    $t \leftarrow t + 1$
end while
```

At the beginning of the loop of generations, *Q(t)* is observed to generate the classical population *C(t)*. As reported before, the user can provide initial probabilities or the program assigns the same probability to all functions, creating a uniform PMF.

The generation t loop with observing quantum individuals generate candidate solutions. The candidate solution comprises a network architecture description and the fixed hyperparameters' values defined previously by the user. It is important to emphasize that each quantum individual can generate one or more classical individuals as long as they generate the same number of individuals. Otherwise, we would favor some quantum individuals over the others. The number of individuals in the classical population $C(t)$ is a multiple of N:

$$C(t) = \left\{c_1^t, c_2^t, \ldots c_{m \cdot N}^t\right\}, \quad m \in \mathbb{N} \tag{4-3}$$

The parameter *repetition (m)* specifies the number of classical individuals per quantum individuals that will be generated. If $repetition = 3$ each quantum individual generates three classical ones.

Once ready, *C(t)* can be evaluated. The evaluation procedure involves training the candidate networks for a few epochs - in this work, just like the previous version, it was defined 50 epochs - with a subset of the training data. Then, following the (Szwarcman 2020) work, from epoch 45 onward, we evaluate the network according to a predefined metric, such as accuracy, using a validation dataset at the end of each epoch. Finally, assign the best of the 5 evaluation results as the individual's fitness. It is calculated the accuracy only at the final five epochs to save time, as it is expensive to stop training and perform evaluations. One should note that the network weights are initialized using the proposed method in (He et al. 2015) which takes ReLU/PReLU into account.

It is important to mention that the algorithm implemented by (Szwarcman 2020) has a timeout flag for these training sessions. If the first 45 epochs of training (before accuracy evaluations start) take more than 90 minutes, the candidate network receives a fitness value of zero. This was implemented in order to eliminate structures that take too long to train, thus creating pressure toward more efficient models.

In the first generation, *C(t)* individuals are ranked and stored in *P(t)*. In generation $t = 0$, they do not have any previous population, so they store in *P(t)* all the individuals they just evaluated. Note that classical recombination is only possible after the first generation. It was also decided not to apply crossover operators in the network architecture, based on the following observation: Blocks of subsequent nodes can repeatedly appear during evolution, and new or unexpected sequences might be discovered. Unlike the first generation, since *P(t)* exists, a new population *C(t)* must be evaluated and then select individuals to be stored.

Consequently, the authors developed Q-NAS to use a steady-state technique, in which they select the k worst individuals from the old population with the k best from the new populations. This method orders the old and new population and keeps the k best individuals (Szwarcman, Civitarese e Vellasco 2019). For example, consider a population of K individuals. Every generation it is created $C(t)$ with size K and then they keep the K best individuals from $[C(t) \cup P(t)]$. The authors also studied that the elitism technique does not outperform the steady-state

Finally, quantum individuals are updated based on the best classical individuals. It is important to emphasize that the update increases the probability of the most promising function, based on its fitness, and consequentially reduces the other probabilities proportionally so that it continues to add up to 1. 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 update step completes the algorithm loop, which is repeated for T generations (Szwarcman 2020). The parameter *update_quantum_gen* is used to establish the frequency in which the update procedure will be conducted.

When the evolution is complete, the final architecture is retrained from scratch for 300 epochs using the fixed hyperparameters and all the available training data. In addition, the network is evaluated every ten epochs using a validation dataset. The periodic evaluations' accuracy is used to save the best model during the retraining phase. When training is over, the best validation model is applied to the test data to obtain the final accuracy value. The test accuracy is used to compare the models among different experiments and with other works (Szwarcman 2020). The following Table 4.1 parameters are used in Q-NAS and are defined by the user:

Parameter Name	**Description**
n_samples	number of samples that will be used during the evolution phase in the algorithm
max_generations	maximum number of generations to run the algorithm
num_quantum_individuals	number of quantum individuals in quantum population
max_num_nodes	maximum number of nodes in the network structure (network size)
penalize_number	maximum number of reducing layers a network can have without suffering penalization
repetition	number of classical individuals each quantum individual will generate
fn_dict	dictionary defining the functions for the network search space and their initial probabilities
update_quantum_gen	periodicity, in generations, in which the quantum individuals will be updated
update_quantum_rate	rate for all the quantum update operations

Table 4.1: Q-NAS algorithm types of Parameters

4.3 Q-NAS Parameters and Previous Experiments

The authors (Szwarcman 2020) made three types of experiments. In the first group, they use the CIFAR-10 benchmark dataset, which contains 60000 colored images of size 32 x 32 pixels, divided into training and test sets – 50000 and 10000 examples, respectively. The images are labeled for ten categories, such as dog, cat, or airplane.

The second set of experiments requires a more challenging dataset, so it was selected CIFAR-100. It has the same properties as CIFAR-10, except for the number of classes that is ten times bigger. Thus, for the 50,000 training examples, CIFAR-100 has only 500 examples per class.

Finally the last experiment involves a seismic image classification task using Q-NAS to solve it.

It is important to notice that the authors made the search space for the network structure and the hyperparameters, which means that they also made a hyperparameter optimization. However, as they stated, the default Tensorflow's hyperparameters outperformed the hyperparameters found by Q-NAS.

In (Szwarcman 2020), two evolutions were implemented for each experiment, with different search spaces: one using functions with convolutional blocks, as shown in Table 4.2, and another using functions with residual blocks as shown in Table 4.3 in the search space and their initial probabilities, respectively.

function name	function	kernel size	stride	filters	initial probability
conv_k_1_f	ConvBlock	[1, 3, 5]	1	[64, 128, 256, 512]	0.028
max_pool_2_2	MaxPool	2	2	-	0.167
avg_pool_2_2	AvgPool	2	2	-	0.167
no_op	NoOp	-	-	-	0.333

Table 4.2: Functions with convolutional blocks (Szwarcman 2020)

function name	function	kernel size	stride	filters	initial probability
bv1_3_1_f	ResidualV1	3	1	[64, 128, 256]	0.055
bv1p_3_1_f	ResidualV1Pr	3	1	[64, 128, 256]	0.055
max_pool_2_2	MaxPool	2	2	-	0.167
avg_pool_2_2	AvgPool	2	2	-	0.167
no_op	NoOp	-	-	-	0.333

Table 4.3: Functions with residual blocks (Szwarcman 2020)

Tables 4.2 and 4.3 show the function sets for pooling, convolutional and residual layers defined in the Q-NAS search space. The idea is that Q-NAS should find the set of layers to form an architecture that achieves the best fitness.

The *ConvBlock* comprises a convolutional layer, batch normalization, and ReLU activation. Zero-padding is also used in the convolution layer input borders. The others are straightforward: the *Pooling* function can be a max-pooling or an average pooling layer; *NoOp* is the no-operation function that allows us to represent variable-length networks.

The Table 4.3 explores residual units instead of convolutional functions. To understand the idea behind residual learning, first consider two networks: a shallow one and a deep counterpart, created by adding identity layers on the smaller model. The authors (He et al. 2016) stated that this construction should indicate that a deeper network could not present higher training errors than its shallow equivalent. Since their experience showed otherwise, they proposed to make the layers fit a residual mapping instead of the original unreferenced mapping. If the stacked identity layers were the optimal solution, they claim it would be easier to zero out the residual than directly try to fit the identity. If we denote the underlying mapping as $H(x)$, the residual mapping is then $F(\boldsymbol{x}) \equiv H(x) - \boldsymbol{x}$. The original mapping becomes $F(x) + x$. The unit's shortcut(or skip connection) can be directly used if the input x and the output $F(x) + x$ have the same dimensions. When the sizes do not match, there are two typical approaches to fix this issue. The first one pads the input with zeros when performing the summation. The second uses a projection shortcut, which applies a 1x1 convolution to fix the dimensions.

Thus, in Q-NAS it was adopted two types of units: the ResidualV1, with identity shortcut, and the ResidualV1Pr, with projection shortcut as described in Table 4.3.

One should notice that in the individual from Table 4.2 the authors decided to list function specifications that are relatively inexpensive concerning computational cost. Note that one can be as general as desired regarding the options of kernel, strides, and number of filters. Furthermore, one can observe that the convolutional layers have a stride of 1, which means that they do not reduce the input size since they apply zero-padding.

The initial probabilities were equally divided between the three types of functions: NoOp received 1/3, and both ConvBlock and Pooling received 1/3 divided by the number of options of each. kind. The same logic was implemented for the Residuals Block.

The hyperparameters used to find the best individual were fixed with the values described in Table 4.4. Moreover the Q-NAS parameter configuration is described in Table 4.5. This parameters were the baseline for the experiments that will be showed in Chapter 7, since it was the set of parameters that was used to find and train the individual that reached the best accuracy.

optmizer	decay	learning rate	momentum	weight decay
RMSProp	0.9	1.0e-3	0.0	1.0e-4

Table 4.4: Fixed Hyperparameters used in Q-NAS

parameter	value
max_generations	300
max_num_nodes	20
penalize_number	3
num_quantum_ind	5
repetition	4
update_quantum_gen	5
update_quantum_rate	0.1

Table 4.5: Q-NAS algorithm parameters

As described before, the first experiment consists in finding the architecture that achieves the highest accuracy in CIFAR-10 dataset. With the parameters described in Tables 4.4 and 4.5, the authors evolved an architecture with Q-NAS using 10000 samples and retraining this same architecture for 300 epochs. In the retraining phase, it is used 60000 samples. Table 4.6 shows the final individual found for the convolutional blocks. The best architecture found in (Szwarcman 2020) was the one evolved with the Residual Blocks search space. They were able to achieve an accuracy of 93.85% for 67 GPU days. This individual is described in Table 4.7. For the evolution with *ConvBlocks* their best accuracy was of 93.70% using early-stopping mechanism.

However, as the authors from the previous version did not defined the architecture found, we decided to use the architecture that was the cited in the previous work and reached an accuracy of 92.95% without using the early-stopping mechanism.

The same experiment was made with CIFAR-100 dataset containing 10000 images in the evolution phase (9000 for training and 1000 for validation). The parameters described in Table 4.5 were the same used in this experiment with the exception of *max_num_nodes* that was 30 instead of 20. The authors (Szwarcman 2020) reached an accuracy of 74.23% with the individual described in the Table 4.8.

Using early-stopping mechanism they evolved for 156 GPU days. Q-NAS could reach accuracy levels comparable to other methods for CIFAR-100, as can be seen in Table 6.17 without any adjustments on parameter values or the early-stopping mechanism, indicating robustness in the algorithm.

One can notice that it is possible to create new networks with much less layers than the one defined in *max_num_nodes.*

One should also note that we did not describe the hyperparameter optimization since the authors from previous version of Q-NAS stated that the hyperparameter optimization did not improve when compared with the

Nodes	Function
0	conv_5_1_512
1	no_op (x2)
2	conv_3_1_128
3	conv_3_1_512
4	no_op (x2)
5	conv_5_1_256
6	avg_pool_2_2
7	no_op (x3)
8	conv_3_1_256
9	avg_pool_2_2
10	no_op
11	conv_5_1_128
12	avg_pool_2_2
13	max_pool_2_2
14	no_op (x2)

Table 4.6: Best architecture with convolutional blocks found in Q-NAS for CIFAR-10 without Early-stopping mechanism.

Nodes	Function
0	bv1p_3_1_128
1	bv1p_3_1_128
2	bv1p_3_1_256
3	avg_pool_2_2
4	no_op
5	bv1p_3_1_256
6	no_op(3x)
7	max_pool_2_2
8	max_pool_2_2
9	bv1_3_1_128
10	bv1_3_1_64
11	bv1p_3_1_256
12	bv1_3_1_256
13	no_op
14	max_pool_2_2
15	bv1_3_1_256
16	bv1p_3_1_64
17	no_op

Table 4.7: Best architecture with residual blocks found in Q-NAS for CIFAR-10.

default values from Tensorflow library.

The motivation of this work involves looking for a more generic architecture in a scalable and fast way so that it can be applied to computers with fewer requirements than the ones described before. As mentioned earlier, QIEA

Nodes	Function
0	bv1_3_1_128
1	bv1p_3_1_64
2	max_pool_2_2
3	bv1_3_1_256
4	bv1p_3_1_128
5	bv1p_3_1_256
6	bv1p_3_1_256
7	avg_pool_2_2
8	bv1p_3_1_256
9	bv1p_3_1_256
10	avg_pool_2_2
11	avg_pool_2_2

Table 4.8: Best architecture found in Q-NAS for CIFAR-100.

can find better solutions with fewer evaluations.

Thus, this work presents Enhanced Q-NAS an algorithm capable to find a more complex structure in a larger search space and using only 4 GPUs and still achieving a competitive accuracy when compared to the related works. In terms of scalability, Enhanced Q-NAS is an improved algorithm that implements a new optimizer and build a network from scratch to apply in CIFAR-100 context. Finally, and most importantly, enhanced Q-NAS is the first QIEA, as far as the authors are concerned, applied to detect COVID-19 in computed tomography chest images.

5
Enhanced Q-NAS

In the previous work (Szwarcman 2020), the authors achieved a competitive accuracy building a network from scratch for CIFAR-100. Since our goal is to enhance the algorithm, due to the need for comparison with the old version of Q-NAS algorithm, this work is also focused on the classification task. Moreover, we want to apply it on a case study in a real medical image dataset.

However, due to the dataset complexity, there was a need to carry out a more in-depth study in order to achieve a higher accuracy. Our main focus will be in the improvements for CIFAR-100 dataset. Moreover, the authors used 20 GPUs to make their experiments that was cited in this work. The focus of this work is to make Q-NAS more scalable and improve the result without making huge changes in the algorithm.

For that, seeking to improve results and reduce processing time this Section presents improvements we made in the (Szwarcman 2020). Initially, we changed the optimizer RMSProp to SWATS algorithm proposed by (Keskar e Socher 2017). Furthermore, an experiment was conducted that evolved a network from scratch, using Q-NAS to CIFAR 100 dataset, just changing a few parameters of the algorithm and transforming it into a problem to be solved with fewer GPUs. The GPUs used in the following experiments were 4 Nvidia GTX 1080. The resultant network is relatively small (20 layers) compared to other state-of-the-art models and achieve promising accuracy with considerably less computational cost than other NAS algorithms.

5.1
Switching ADAM to SGD Optimizer

During the training in image classification, we want to adjust the network weights so that the highest score appears for the correct label of the corresponding image. We must define an error function to measure how far the current output is from the correct answer. The learning algorithm minimizes this error function (or cost function) concerning the network's weights. Stochastic Gradient Descent (SGD) and its variants are the most popular optimization algorithms for training CNNs. First, it presents a *batch* of input arrays - images, then it computes the output scores and errors. Sequentially, it is calculated the average gradients for the batch examples. Finally, the weights are adjusted accordingly. The learning rate is a critical parameter for the SGD algorithm, as it can affect the model performance significantly.

The original Q-NAS (Szwarcman 2020) used RMSProp optimizer during the evolution phase to find the best architecture and the SGD for the re-training phase (Tieleman, Hinton et al. 2012).

RMSProp is an adaptive method which diagonally scale the gradient via estimates of the function's curvature. This method can be interpreted as a method that use a vector of learning rates, one for each parameter, that are adapted as the training algorithm progresses. It also discards contributions from the extreme past with the help of an exponentially decaying moving average. RMSProp adds the moving average decay to the list of parameters to be specified by the user. However, as stated by (Wilson et al. 2017), RMSProp, just like the optimizer Adam, tends to be insufficient at generalizing in a fashion comparable to SGD. These methods tend to perform well in the initial portion of training but are outperformed by SGD at later stages of training.

With this motivation, the authors (Keskar e Socher 2017) proposed the Switch ADAM to SGD (SWATS) algorithm. The switch is designed to be automatic and one does not introduce any more hyperparameters. The switchover point and the SGD learning rate are both learned as a part of the training process. A projection of the Adam step is monitored on the gradient subspace and use its exponential average as an estimate for the SGD learning rate after the switchover.

Adaptive algorithms, such as Adam, converge fast and are suitable for processing sparse data. SGD with momentum can converge to more accurate results. The combination of SGD and Adam develops the advantages of both methods. Specifically, it first trains with Adam to quickly drops the loss and then switches to SGD for precise optimization based on the previous parameters at an appropriate switch point (Sun et al. 2019).

However, there are two issues to point in this operation: the first is when to switch from Adam to SGD and the second is how to adjust the learning rate after switching the optimization algorithm. To solve this, the authors (Keskar e Socher 2017) proposed the following theory: the movement d^{Adam} of the parameter at iteration t of the Adam is:

$$d_t^{Adam} = \frac{\eta^{Adam}}{V_t} m_t \tag{5-1}$$

where η^{Adam} is the learning rate of Adam, V_t is the accumulated historical gradient of the parameter and m_t is the decaying average of the gradients.

The movement d^{SGD} of the parameter at iteration t of the SGD is:

$$d_t^{SGD} = \eta^{SGD} g_t \tag{5-2}$$

where η^{SGD} is the learning rate of SGD and g_t is the gradient of the current position. The movement of SGD can be decomposed into the learning rates along Adam's direction and its orthogonal direction. At the same time, SWATS also adjusts its optimized trajectory by moving in the orthogonal direction. Then, we have:

$$\mathrm{Proj}_{\mathrm{Adam}}\, d_t^{SGD} = d_t^{\mathrm{Adam}} \tag{5-3}$$

and derive solution:

$$\eta_t^{SGD} = \frac{\left(d_t^{\mathrm{Adam}}\right)^{\mathrm{T}} d_t^{\mathrm{Adam}}}{\left(d_t^{\mathrm{Adam}}\right)^{\mathrm{T}} g_t} \tag{5-4}$$

where $\mathrm{Proj}_{\mathrm{Adam}}$ is the projection in the direction of Adam. To reduce noise, a moving average can be used to correct the estimate of the learning rate,

$$\lambda_t^{SGD} = \beta_2 \lambda_{t-1}^{SGD} + (1-\beta_2)\, \eta_t^{SGD} \tag{5-5}$$

$$\tilde{\lambda}_t^{SGD} = \frac{\lambda_t^{SGD}}{1-\beta_2} \tag{5-6}$$

where λ_t^{SGD} is the first moment of learning rate η^{SGD}, and $\tilde{\lambda}_t^{SGD}$ is the learning rate of SGD after converting and β_2 is exponential decay rate. For switch point, a simple guideline $\left|\tilde{\lambda}_t^{SGD} - \lambda_t^{SGD}\right| < \epsilon$ is often used.

5.2 Cyclic Learning Rate

Another approach which was experimented in this work was making the learning rate from RMSProp optimizer cyclical. It is well known that a too small learning rate will make a training algorithm converge slowly while a too large learning rate will make the training algorithm diverge (Orr e Müller 1998). The authors (Smith 2017) stated that increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect.

This motivated the authors to design the learning rate to vary within a range of values rather than adopting a stepwise fixed or exponentially decreasing value. This means that one could define a maximum and minimum boundaries and the learning rate cyclically varies between these bounds. A more practical reason as to why Cyclic Learning Rate (CLR) works is that it is likely the optimum learning rate will be between the bounds and near optimal learning rates will be used throughout training.

The authors designed a triangular method which linearly increase and linearly decrease the learning rate. The red curve in Figure 5.1 shows the result of the triangular policy on CIFAR-10. The settings used to create the red curve were a minimum learning rate of 0.001 (as in the original parameter file) and a maximum of 0.006.

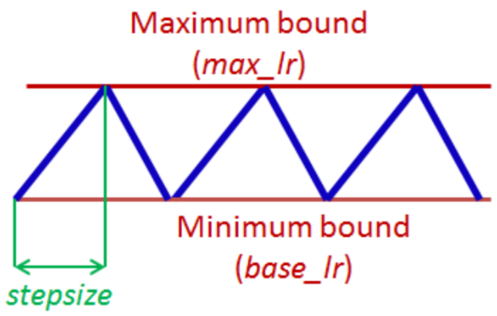

Figure 5.1: Cyclic Learning Rate triangular method (Smith 2017)

The general schedule can be described as:

$$\eta_t = \eta_{min} + (\eta_{max} - \eta_{min})(max(0, 1 - x)) \tag{5-7}$$

where x is defined as:

$$x = \left| \frac{\text{iterations}}{\text{stepsize}} - 2(\text{ cycle }) + 1 \right| \tag{5-8}$$

and cycle can be calculated as:

$$\text{cycle} \;=\; \text{floor}\left(1 + \frac{\text{iterations}}{2\ (\text{stepsize})}\right) \tag{5-9}$$

where η_{min} and η_{max} define the bounds of our learning rate, iterations represents the number of completed mini-batches, stepsize defines one half of a cycle length.

6
Experiments

6.1
Switching ADAM to SGD for CIFAR-10 and CIFAR-100 Improvements

The experiments conducted by previous Q-NAS work (Szwarcman 2020) showed that it was possible to achieve the accuracy of 93.85% for CIFAR-10 and 74.23% for CIFAR-100 datasets evolving an architecture with Q-NAS using 10000 samples in each experiment and retraining the final architectures for 300 epochs. In the retraining phase, it is used 60000 samples.

Two evolutions were implemented, with different search spaces: one using functions with convolutional blocks, as shown in Table 4.2, and another using functions with residual blocks as shown in Table 4.3 in the search space. Finally, the architecture with residual layers was the one with the best accuracy. Table 4.7 presents the best architecture for CIFAR-10 dataset. This work also updates the Q-NAS algorithm to run in Tensorflow 2.3.

Additionally, they used RMSProp optimizer in the evolution and SGD in the retraining phase (Hinton, Srivastava e Swersky 2012) with Tensorflow's default hyper-parameters: *decay = 0.9*, *learning rate= 1.0e-3*, *momentum = 0.0*, *weight decay = 1.0e-4*. Since Tensorflow's default hyperparameters outperform the hyperparameter evolution in Q-NAS, as described before, it is not the interest of this work to present this evolution.

This work presents a new enhancement for Q-NAS algorithm, which is to change RMSProp to SWATS optimizer, proposed by (Keskar e Socher 2017) as described in this section. Our first experiment was to use SWATS algorithm in the training during the evolution phase, just like the authors (Szwarcman 2020) used RMSProp. However, not only increased the time of each iteration in the evolution but it also did not showed any improvements in the fitness value. Thus, since applying SWATS in the Q-NAS evolution phase showed no better performance than using RMSProp, this work proposes to apply SWATS only in the retraining phase.

In order to achieve better accuracy and compare the results, the best architectures found in (Szwarcman 2020) for CIFAR-10 were selected. This architectures were defined in Tables 4.6 and 4.7, using convolutional blocks and residual blocks, respectively. The idea is that Q-NAS should find the set of layers to form an architecture that outperforms the previously fitness and accuracy.

For this experiment, we test different types of learning rates for SWATS that it is described in Tables 6.1 and 6.2. As stated by (Keskar e Socher 2017), this changes in learning rate procedure to perform better than a generic grid-search or hyperparameter optimization, given the vastly different scales of learning rates needed for different modalities. We also fixed the β_1, β_2 and ϵ values, following the (Keskar e Socher 2017) experiments.

It can be noticed that in all cases the SWATS-1 outperforms the other configurations. In CIFAR-10, the higher the learning rate values, the lower are the accuracies.

configuration	β_1	β_2	ϵ	learning rate	accuracy
SWATS-1	0.900	0.999	1.0e-9	0.001	93.39%
SWATS-2	0.900	0.999	1.0e-9	0.003	91.09%
SWATS-3	0.900	0.999	1.0e-9	0.005	89.43%

Table 6.1: SWATS results with differents learning rates for CIFAR-10 individual with Convolutional Layers.

configuration	β_1	β_2	ϵ	learning rate	accuracy
SWATS-1	0.900	0.999	1.0e-9	0.001	94.85%
SWATS-2	0.900	0.999	1.0e-9	0.003	92.21%
SWATS-3	0.900	0.999	1.0e-9	0.005	91.13%

Table 6.2: SWATS results with differents learning rates for CIFAR-10 individual with Residual Layers.

When this network was retrained using SWATS optimizer, we achieved an accuracy of 94.95% for CIFAR-10, which surpassed the previous work (Szwarcman 2020) by 1.10 percentual points. In this specific case, the GPU/-days information was not considered, as only the retraining phase was changed. Based on these results, we can conclude that by only changing the optimizer RMSProp to SWATS, we were able to achieve a new higher score for Q-NAS in CIFAR-10 context. The comparative results with other works are described in Table 6.3 for CIFAR-10.

One should note that there are some algorithms from chapter 2 that are not presented in the Table. This is because not all algorithms can be directly compared to Q-NAS since they have different search space from what it is proposed and different environments (i.e. Federated Learning environments, use of skip connections).

Hand-designed models	accuracy(%)	#params	GPU days
ResNet (He et al. 2016)	93.57	1.7M	-
VGG (Simonyan e Zisserman 2014)	92.06	15.2M	-
GoogleLeNet (Szegedy et al. 2015)	93.64	-	-
NAS			
Meta-QNN (Baker et al. 2016)	93.08	11.18M	100
DARTS (Liu, Simonyan e Yang 2018)	97.24	3.3M	5
NAS-Net (Zoph et al. 2018)	96.86	3.3M	2000
Block-QNN-S (Zhong et al. 2018)	96.46*	39.8M	96
Q-NAS (Szwarcman 2020)	93.85	7.07M	67
Q-NAS Enhanced	94.95	3.6M	67

Table 6.3: Comparing our results with some literature models. The '*' marks the methods that used other datasets for the search and applied the network on CIFAR-10.

6.2 Cyclic Learning Rate

Before estimating a good value for the cycle length as stated by the authors in (Smith 2017), one should note that an epoch is calculated by dividing the number of training images by the batch size used. Q-NAS used a batch size of 256 for 60000 samples in CIFAR-10. Thus, an epoch = 60000/256 = 234 iterations. The authors stated that it is often good to set stepsize equal to 2 to 10 times the number of iterations in an epoch. In this work, we decided to set different stepsizes of 2, 3, 4 and 5 x *epoch*. The result accuracy for CIFAR-10 dataset is described in Tables 6.4 and 6.5, while the result accuracy for CIFAR-100 dataset is described in Tables 6.6 and 6.7.

base learning rate	maximum learning rate	stepsize	accuracy
0.001	0.006	2 x epoch	87.43%
0.001	0.006	3 x epoch	87.09%
0.001	0.006	4 x epoch	88.26%
0.001	0.006	5 x epoch	88.11%

Table 6.4: Accuracies for different stepsizes in CIFAR-10 Residual Individual

From these Tables we could conclude that using cyclic learning rate with the triangular method in all cases did not outperform the baseline results from (Szwarcman 2020). However, we did not tested other methods reported by the authors (Smith 2017) such as *triangular2*, which is the same as the triangular policy, except the learning rate difference is cut in half at the end of each cycle, or *exp_range*, which the learning rate varies between the minimum and

base learning rate	maximum learning rate	stepsize	accuracy
0.001	0.006	2 x epoch	85.22%
0.001	0.006	3 x epoch	86.86%
0.001	0.006	4 x epoch	86.10%
0.001	0.006	5 x epoch	85.09%

Table 6.5: Accuracies for different stepsizes in CIFAR-10 Convolutional Individual

base learning rate	maximum learning rate	stepsize	accuracy
0.001	0.006	2 x epoch	68.13%
0.001	0.006	3 x epoch	67.88%
0.001	0.006	4 x epoch	68.01%
0.001	0.006	5 x epoch	68.24%

Table 6.6: Accuracies for different stepsizes in CIFAR-100 Residual Individual

base learning rate	maximum learning rate	stepsize	accuracy
0.001	0.006	2 x epoch	63.32%
0.001	0.006	3 x epoch	64.68%
0.001	0.006	4 x epoch	64.02%
0.001	0.006	5 x epoch	64.00%

Table 6.7: Accuracies for different stepsizes in CIFAR-100 Convolutional Individual

maximum boundaries and each boundary value declines by an exponential factor of $gamma^{iteration}$. This could be a possible test for future works.

6.3 Evolving a Network from Scratch for CIFAR-100 Classification Task

Another experiment was explored in Q-NAS algorithm: to achieve a higher score in the CIFAR-100 dataset without using the CIFAR-10 evolved network architecture. In (Szwarcman 2020), the authors achieved the best accuracy for CIFAR-100 of 74.23% in their single experiment, without using CIFAR-10 best individual architecture. In the last Q-NAS version, the authors evolved CIFAR-100, from scratch, with 300 generations, 50 epochs per generation, a maximum number of layers of 30, a maximum number of pooling layers of 3 and number of samples to be used in the evolution phase of 10000 as it is described in Table 6.8. Note that the main difference between the CIFAR-10 parameter configuration is the *max_num_nodes* that changed from 20 to 30. This is because CIFAR-100 is a dataset with much more classes than CIFAR-10. Thus, the authors decide to increase the network complexity to achieve a competitive result.

However, the best individual found in (Szwarcman 2020) for CIFAR-100 has less than 20 layers. Because of this, we decided to decrease the maximum number of nodes to 20 in the following experiments.

It is highly important to note is that the previous experiment was made using 20 GPUs. Our main goal is to reach a higher accuracy using only 4 GPUs, which can make the evolution more scalable.

parameter	value
num_samples	10,000
max_generations	300
max_num_nodes	30
penalize_number	3
num_quantum_ind	5
repetition	4
update_quantum_gen	5
update_quantum_rate	0.1

Table 6.8: Q-NAS algorithm parameters for CIFAR-100 previous Q-NAS version experiment

Our first experiment involves reducing the maximum number of nodes from 30 to 20 again and reducing the maximum number of generations to 100. Moreover, we doubled the number of samples to use during the evolution phase, which means that we have now 20000 samples. Finally, since we are using only 4 GPUs, we decreased the number of quantum individuals and the number of classical individuals that can be generated by the quantum one. Thus, our parameter table is described in 6.9. The right arrow indicates the changes in parameter configuration with the baseline experiment from (Szwarcman 2020). It is important to emphasize that the previous version of Q-NAS tested the usage of 4 classical individuals but not the setting with number of quantum individuals of 2 and number of classical individuals generated by each quantum individuals of 2.

The first result found with this configuration was an accuracy of 70.74% for 16 GPU days for the best individual (Table 6.10), which overcame the first value found by (Szwarcman 2020) which were 69.95%. Even though the residual networks are slower to train, the fact that residual networks allow other forms of learning with layer bypass helps to make more flexible combinations which is reflected in the obtained results. The best individuals can be seen in Tables 6.11 and 6.12, considering convolutional and residual individuals, respectively. It is important to note that the main difference was the number of samples used. Still, the training did not take much longer because we are

parameter	value
num_samples	10,000 $\rightarrow 20,000$
max_generations	300 $\rightarrow$ 100
max_num_nodes	30 $\rightarrow$ 20
penalize_number	3
num_quantum_ind	5 $\rightarrow$ 2
repetition	4 $\rightarrow$ 2
update_quantum_gen	5
update_quantum_rate	0.1

Table 6.9: Main changes in Q-NAS parameterization to evolve CIFAR-100. The value before the arrow represents the value from the previous Q-NAS version.

only using 2 quantum individuals to generate 2 classical individuals. Because of this, we increased the number of samples in the evolution phase in the following experiments.

Individual	fitness	best accuracy in validation set	final test accuracy	GPU days
Convolutional	58.20%	63.12%	62.44%	12.8
Residual	64.21%	70.93%	70.74%	16

Table 6.10: First Experiment Result for CIFAR-100 dataset from the Individuals found that are described in Tables 6.11 and 6.12

Nodes	Function
0	conv_5_1_128
1	max_pool_2_2
2	conv_1_1_512
3	conv_5_1_256
4	conv_1_1_512
5	conv_5_1_128
6	conv_5_1_128
7	avg_pool_2_2
8	conv_5_1_256
9	avg_pool_2_2
10	no_op
11	conv_5_1_256
12	conv_1_1_128
13	no_op

Table 6.11: First Experiment Convolutional Individual

Finally, we decided to increase once more the dataset samples from 20000 to 35000 and decrease the total number of generations to 200. All parameters for this experiment are described in Table 6.13. With these settings we were

Nodes	Function
0	no_op
1	bv1_3_1_64
2	bv1_3_1_64
3	no_op
4	no_op
5	bv1p_3_1_64
6	no_op (x2)
7	bv1p_3_1_256
8	avg_pool_2_2
9	bv1p_3_1_256
10	bv1p_3_1_256
11	avg_pool_2_2
12	bv1p_3_1_128
13	no_op
14	bv1_3_1_64
15	avg_pool_2_2

Table 6.12: First Experiment Residual Individual

able to reach our final accuracy of 76.39% for 18 GPU days, outperforming the evolution by 2.16 from the evolution from scratch in CIFAR-100.

The results can be seen in Table 6.14. It is possible to observe that the residual individual outperforms the convolutional as it is expected. The final network architecture is described in Table 6.16. The comparative results with other works are presented in Table 6.17.

From these results, we can conclude that increasing the number of samples might be a good alternative for getting a higher accuracy without increasing the number of GPU days used. As a result, we were able to achieve a higher accuracy using only four GPUs which shows that Q-NAS can be more scalable. Moreover, we were able to find an architecture that outperforms the previous work using the maximum number of nodes of twenty and with only two quantum individuals. Finally, the residual network outperforms the convolutional just like the (Szwarcman 2020) work.

parameter	value
num_samples	10,000 $\rightarrow$ 35,000
max_generations	300 $\rightarrow$ 200
max_num_nodes	30 $\rightarrow$ 20
penalize_number	3
num_quantum_ind	5 $\rightarrow$ 2
repetition	4 $\rightarrow$ 2
update_quantum_gen	5
update_quantum_rate	0.1

Table 6.13: Main changes in Q-NAS parameterization to evolve CIFAR-100. The value before the arrow represents the value from previous Q-NAS version.

Individual	fitness	best accuracy in validation set	final test accuracy	GPU days
Convolutional	68.3%	72.6%	72.1%	14.1
Residual	69.40%	76.8%	76.39%	18

Table 6.14: Final Experiment Result for CIFAR-100 dataset from the Individuals found that are described in Tables 6.15 and 6.16. They were the best individuals that outperforms the previous work by (Szwarcman 2020).

Node	Function
0	conv_5_1_32
1	conv_3_1_64
2	no_op
3	max_pool_2_2
4	max_pool_2_2
5	conv_1_1_64
6	no_op
7	conv_3_1_256
8	no_op
9	conv_3_1_64
10	conv_5_1_256
11	conv_1_1_32
12	conv_3_1_64
13	conv_5_1_256
14	conv_5_1_64
15	no_op
16	avg_pool_2_2
17	conv_3_1_256
18	conv_5_1_64
19	conv_3_1_64

Table 6.15: Best Architecture with Convolutional Blocks found in **Q-NAS Enhanced** for CIFAR-100

Nodes	Function Names
0	bv1_3_1_64
1	bv1p_3_1_64
2	avg_pool_2_2
3	bv1_3_1_64
4	bv1_3_1_256
5	max_pool_2_2
6	bv1p_3_1_256
7	bv1p_3_1_128
8	no_op
9	bv1_3_1_256
10	bv1_3_1_64
11	bv1_3_1_64
12	no_op
13	bv1_3_1_64
14	no_op
15	no_op
16	bv1p_3_1_64
17	no_op
18	bv1p_3_1_128
19	bv1_3_1_256

Table 6.16: Best Architecture with Residual Blocks found in **Q-NAS Enhanced** for CIFAR-100

Hand-designed models			
	accuracy(%)	#params	GPU days
ResNet-1001 (He et al. 2016)	77.30	10.2M	-
ResNet-164 (He et al. 2016)	75.67	1.7M	-
Network in Network (NiN) (Lin, Chen e Yan 2013)	64.32	-	-
NAS			
Meta-QNN (Baker et al. 2016)	72.86*	11.18M	100
Block-QNN-S (Zhong et al. 2018)	81.94	39.8M	96
Q-NAS (Szwarcman 2020)	74.23	6.25M	67
Q-NAS Enhanced	76.39	3.8M	18

Table 6.17: Results from the literature on CIFAR-100. The '*' marks the methods that used other datasets for the search and applied the network on CIFAR-100.

6.4 Case Study: COVID-19 Detection in Computed Tomography Images

In this section, we present the use of Q-NAS to evolve an architecture for a real case study involving COVID-19 detection in computed tomography images that outperformed benchmark networks.

The computed tomography chest images used in this work were extracted from patients hospitalized that totalled more than 1500 images. The dataset was divided into two classes: 827 labeled COVID images and 850 labeled healthy images. Each CT image was evaluated and labeled by three experienced radiologists. It is emphasized that the identification of each patient has been eliminated. During the training phase, the dataset was split into 80% training, 10% validation and 10% test.

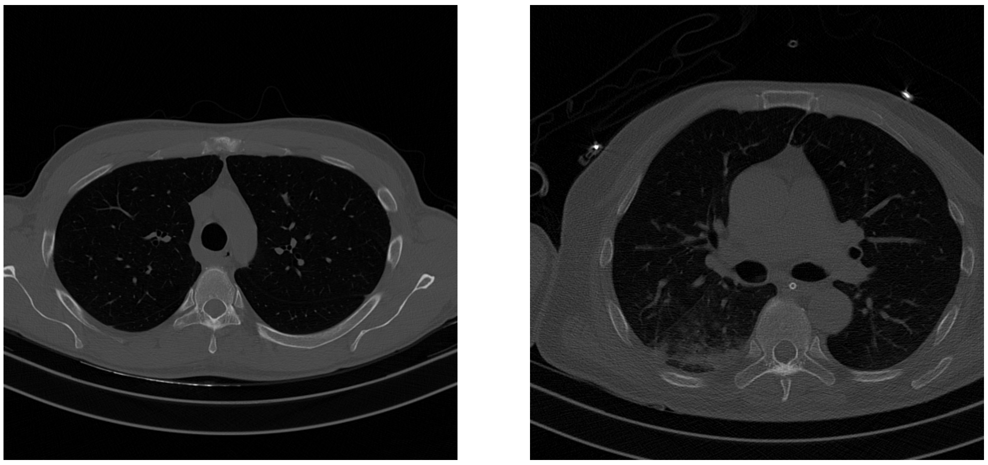
(a) Non-COVID CT (b) COVID CT

Figure 6.1: Computed Tomography Examples

Figure 6.1 shows examples of the two types of computed tomography. The first represents the healthy chest CT, since there are no changes in the lung parenchyma, typical of the presence of the virus, and the second represents the COVID infected chest CT. All the input images were resized to 128-by-128 and each training image was augmented with random cropping, with a scale of 0.5, horizontal flip, random contrast, and random brightness with a factor of 0.2.

Before evolving a Q-NAS architecture, we trained these images in four benchmark networks: VGG, EfficientNet, GoogleLeNet and COVID-Net. Furthermore, the dataset was also used to train the most recent network proposed by (Wang, Lin e Wong 2020), COVID-Net. The hyperparameters values were chosen as Tensorflow's default. In the next sections, we will briefly discuss the architecture of the benchmarks networks.

6.4.1 VGG - Very Deep Convolutional Neural Network

VGG (Simonyan e Zisserman 2014) is a Convolutional Neural Network architecture proposed in 2014. The original idea is that instead of having a large number of hyperparameters they focused on having convolution layers of 3x3 filters with a stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2. In this way, it can simulate larger filters, keeping the benefits of being a small filter. In addition, the small-size convolution filters allows VGG to have a large number of weight layers; of course, more layers leads to improved performance.

The main disadvantage is as the size of the input volume decreases (because of Convolution and Pooling), the depth of the Network increases due to an increase in the number of filters to be applied. The VGG 16 architecture consists of 13 convolutional layers and 3 pooling layers. Figure 6.2 shows the complete architecture of VGG.

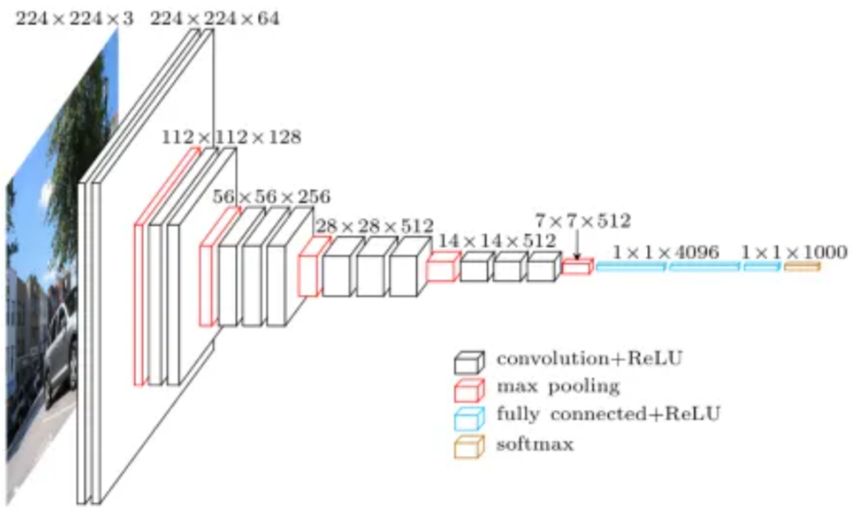

Figure 6.2: VGG Architecture (Simonyan e Zisserman 2014)

6.4.2 EfficientNet

EfficientNet (Tan e Le 2019) network focus on how to scale Convolutional Neural Networks efficiently. (Tan e Le 2019) used a compound scaling method: they simply scale each network dimension by a constant ratio to balance all dimensions of network width/depth/resolution. Unlike conventional practice that arbitrary scales these factors, the EfficientNet method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients.

To explain this compound scaling method, let us have this example: if we want to use 2^N times more computational resources, then we can simply increase the network depth by α^N, width by β^N and image size by γ^N where α, β, γ are constant coefficients determined by a small grid search on the original small model. EfficientNet uses a compound coefficient ϕ to uniformly scale network width, depth, and resolution in a principled way.

The compound scaling method is justified by the intuition that if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image. The resulting architecture uses mobile inverted bottleneck convolution (MBConv). Figure 6.3 shows the complete architecture of EfficientNet.

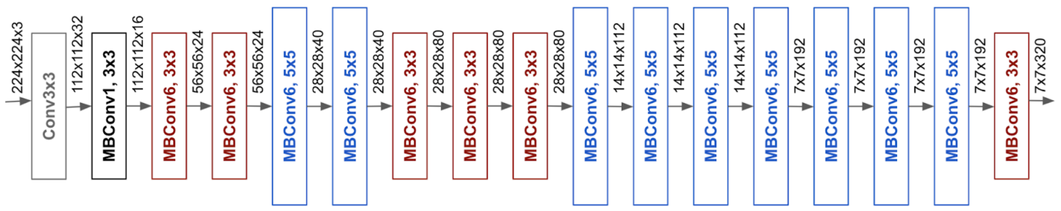

Figure 6.3: EfficientNet Architecture (Tan e Le 2019)

6.4.3 GoogleLeNet

In the GoogleLeNet network (Szegedy et al. 2015), it is proposed a new module called inception. In order to make deep neural networks less computationally expensive, the work presented in (Szegedy et al. 2015) limits the number of input channels by adding an extra 1x1 convolution before the 3x3 and 5x5 convolutions.

An Inception Module consists of the following components: Input layer, 1x1 convolution layer, 3x3 convolution layer, 5x5 convolution layer, Max pooling layer, Concatenation layer. Within an Inception module, they add padding(same) to the max-pooling layer to ensure it maintains the height and width as the other outputs(feature maps) of the convolutional layers within the same Inception module. By doing this, they ensure they can concatenate the outputs of the max-pooling layer with the outputs of the conv layers within the concatenation layer. Figure 6.4 shows the inception module introduced in GoogleLeNet architecutre.

GoogLeNet has nine such inception modules stacked linearly. It is 22 layers deep (27, including the pooling layers). It uses global average pooling at the end of the last inception module. Needless to say, it is a pretty deep classifier. As with any very deep network, it is subject to the vanishing gradient problem.

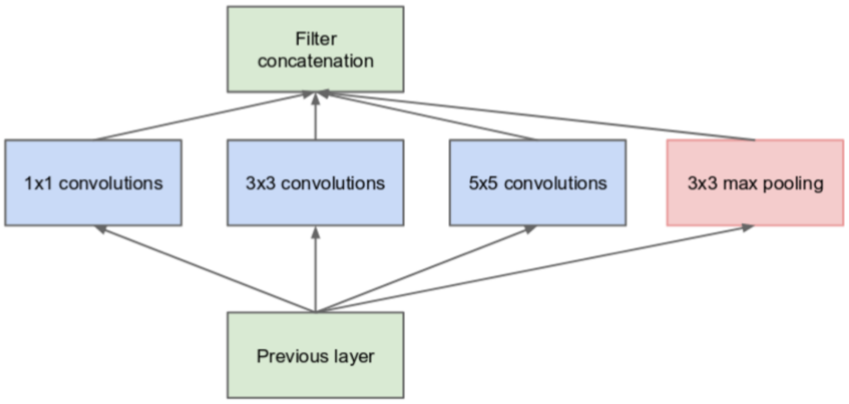

Figure 6.4: Inception Module Architecture (Szegedy et al. 2015)

6.4.4 COVID-Net

COVID-Net, a deep convolutional neural network design tailored for the detection of COVID-19 cases from chest X-ray (CXR) images that is open source and available to the general public (Wang, Lin e Wong 2020). A few points should be considered by the authors during the manual design of COVID-Net architecture:

It can be observed that the COVID-Net network architecture makes heavy use of a lightweight residual projection-expansion projection-extension (PEPX) design pattern, which consists of: 1×1 convolutions for projecting input features to a lower dimension, 1×1 convolutions for expanding features to a higher dimension that is different from that of the input features, efficient 3×3 depth-wise convolutions for learning spatial characteristics to minimize computational complexity while preserving representational capacity, 1×1 convolutions for projecting features back to a lower dimension, and 1×1 convolutions that finally extend channel dimensionality to a higher dimension to produce the final features.

The proposed COVID-Net was pretrained on the ImageNet dataset and then trained on the COVIDx dataset using the Adam optimizer using a learning rate policy where the learning rate decreases when learning stagnates for a period of time (i.e., 'patience'). The following hyperparameters were used for training: learning rate=2e-4, number of epochs=22, batch size=64, factor=0.7, patience=5. Furthermore, data augmentation was leveraged with the following augmentation types: translation, rotation, horizontal flip, zoom, and intensity shift.

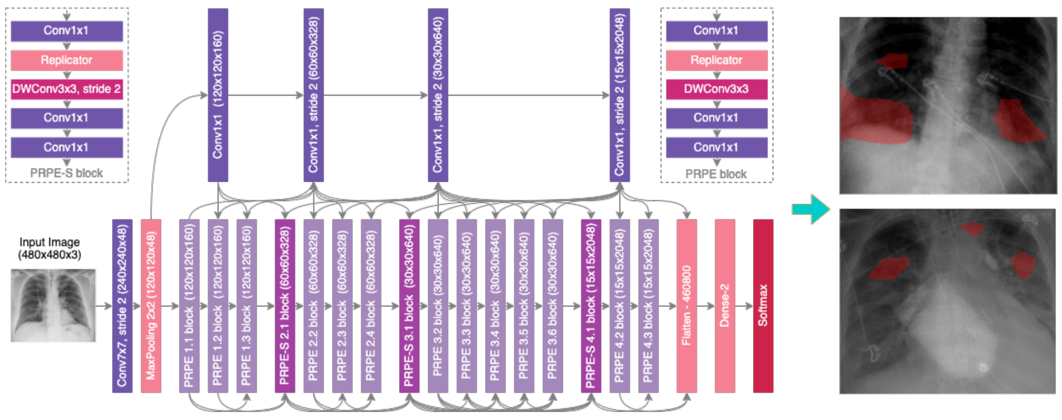
Conv1x1
Replicator
DWConv3x3, stride 2
Conv1x1
Conv1x1
PRPE-S block
Conv1x1 (120x120x160)
Conv1x1, stride 2 (60x60x328)
Conv1x1, stride 2 (30x30x640)
Conv1x1, stride 2 (15x15x2048)
Conv1x1
Replicator
DWConv3x3
Conv1x1
Conv1x1
PRPE block
Input Image (480x480x3)
Conv7x7, stride 2 (240x240x48)
MaxPooling 2x2 (120x120x48)
PRPE 1.1 block (120x120x160)
PRPE 1.2 block (120x120x160)
PRPE 1.3 block (120x120x160)
PRPE-S 2.1 block (60x60x328)
PRPE 2.2 block (60x60x328)
PRPE 2.3 block (60x60x328)
PRPE 2.4 block (60x60x328)
PRPE-S 3.1 block (30x30x640)
PRPE 3.2 block (30x30x640)
PRPE 3.3 block (30x30x640)
PRPE 3.4 block (30x30x640)
PRPE 3.5 block (30x30x640)
PRPE 3.6 block (30x30x640)
PRPE-S 4.1 block (15x15x2048)
PRPE 4.2 block (15x15x2048)
PRPE 4.3 block (15x15x2048)
Flatten - 460800
Dense-2
Softmax

Figure 6.5: COVID-Net Architecture (Wang, Lin e Wong 2020)

In particular, the COVID-Net network architecture is comprised of a heterogeneous mix of convolution layers with a diversity of kernel sizes (ranging from 7×7 to 1×1), and grouping configurations (ranging from ungrouped to depth-wise). The considerable architectural diversity exhibited by the COVID-Net architecture further reinforces the fact that the machine-driven design exploration strategy has tailored the network architecture to a very fine level of granularity for COVID-19 case detection from CXR images to achieve strong representational capacity for a specific task.

6.4.5 Q-NAS experiment and Comparative Methods

For Q-NAS evolution scheme, we defined the following setting to evolve the CT images: 100 generations, 50 epochs per generation, a maximum number of layers of 20, and a maximum number of pooling layers of 8. For the evolution phase, we used the RMSProp optimizer and, for the retraining phase, we used the SWATS optimizer with the same hyperparameters defined in the other experiments. After the evolutionary process, the best architecture found was retrained for 100 epochs. With these settings, we were able to reach the final accuracy of 99.44% for 9 GPU days. The final network architecture can be seen at Table 6.19. Our network contains only 15 layers, which is a competitive number compared to GoogleLeNet which contains 22, VGG-16 that contains 16 layers and EfficientNet which contains 237 layers. One should note that we did not use ResNet's Q-NAS configuration since it is a heavier network and we already achieved a high accuracy, but for future experiments this can be an option. COVID-Net can not be compared in this context because they introduced a new layer design called projection-expansion-projection-extension (PEPX) (Wang, Lin e Wong 2020).

The benchmarks networks (VGG-16, EfficientNet, GoogleLeNet and

parameter	value
max_generations	100
max_num_nodes	20
penalize_number	8
num_quantum_ind	2
repetition	2
update_quantum_gen	5
update_quantum_rate	0.1

Table 6.18: Q-NAS parameters evolution for COVID dataset

Nodes	Function Names
0	no_op
1	no_op
2	no_op
3	avg_pool_2_2
4	conv_3_1_128
5	conv_3_1_128
6	avg_pool_2_2
7	conv_1_1_32
8	conv_1_1_32
9	conv_1_1_64
10	conv_3_1_64
11	avg_pool_2_2
12	avg_pool_2_2
13	conv_3_1_256
14	conv_3_1_256
15	conv_3_1_32
16	conv_5_1_64
17	conv_5_1_32
18	no_op
19	max_pool_2_2

Table 6.19: Best Architecture found for COVID Classification Task

COVID-Net) were trained in the same dataset for 100 epochs using ADAM optimizer. One of the future investigations is to use a SWATS optimizer in these networks to check if they can achieve a higher accuracy. All these networks were pre-trained with the ImageNet dataset, including COVID-Net. We then trained all these networks in our dataset, contemplating all layers. The resulting accuracies can be seen in Table 6.20 where it can be perceived that Q-NAS outperforms the three benchmarks and COVID-Net networks.

Network	Accuracy	Number of Layers
VGG16	92.86%	16
EfficientNetB0	98.25%	237
GoogleLeNet	96.97%	22
COVID-Net	95.88%	-
Q-NAS	**99.44%**	**15**

Table 6.20: Final Accuracy for each Network

7
Conclusion

In this work, we revisited Q-NAS: a quantum-inspired algorithm to search for deep neural network structures. The first thing that we considered is to not optimize the hyperparameters. The motivation behind this is that Q-NAS as reported by (Szwarcman 2020) did not show better results than the default's parameters of Tensorflow. Because of this we decided to focus only in the architecture optimization. We also decided not to make the network search space more complex since we want to find alternatives to evolve a network with only 4 GPUs instead of 20.

We were able to enhance the algorithm settings to improve CIFAR-10 and CIFAR-100 results, when compared to (Szwarcman 2020), using fewer GPUs. The first experiment demonstrated that, by using the SWATS algorithm to retrain the best individual, the original results found in (Szwarcman 2020) could be improved. We make multiples experiments with different learning rates as recomended by the author (Keskar e Socher 2017). The learning rate with the best result was the one with 0.001.

Additionally we provide another experiment using Cyclic Learning Rate for RMSProp optimizer. We chose the triangular method as reported before. As related by the authors, it is often good to set stepsize equal to 2 to 10 times the number of iterations in an epoch. Because of this, we decided to change different configurations of stepsize. However, none of the results found outperforms the baseline work. For future works we intend to test different types of methods such as *triangular2* and $gamma^{iteration}$.

Furthermore, this work also evolved and trained a new architecture from scratch for CIFAR-100 image classification. Instead of increasing the number of layers it was decided to increase the number of samples to be used in the evolution phase and decrease the number of generations. We were able to find an architecture that reached an accuracy of 76.39% with only 18 GPU days with an architecture that contains only 20 layers. With this accuracy we were able to outperform the baseline work using fewer GPUs.

Finally, the enhanced Q-NAS was applied to a real case study to detect COVID-19 in computed tomography chest images. We were able to find a network that achieved an accuracy of 99.44% for just 9 GPU days. As far as we are concerned, this is the first QIEA applied to COVID-19 detection. We were also able to outperform the benchmark networks.

Future works involve using Q-NAS to evolve a network capable of classifying different types of pulmonary diseases in CT chest images. Moreover, more complex functions, with skip connections as well as encapsulated functions, can be added to the search space of the evolutionary process, to include some sequence of layers that frequently appear in the structures from the literature. Finally, we plan to extend the parameters analysis, studying their impact when working with other datasets.

In addition, we could also make an hyperparameter optimization focused on COVID-19 detection extending our parameter analysis and studying their impact for CT images. Another approach that can be explored is the use of Q-NAS for Segmantation or Object Detection task. Since our search space can be modified with different functions, it is possible to use this algorithm to search for different architectures focused on this task.

8
Bibliography

Abadi et al. 2015 ABADI, M. et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: <http://tensorflow.org/>.

Ahmed e Torresani 2018 AHMED, K.; TORRESANI, L. Maskconnect: Connectivity learning by gradient descent. In: **Proceedings of the European Conference on Computer Vision (ECCV)**. [S.l.: s.n.], 2018. p. 349–365.

Awad, Mallik e Hutter 2020 AWAD, N.; MALLIK, N.; HUTTER, F. Differential evolution for neural architecture search. **arXiv preprint arXiv:2012.06400**, 2020.

Bagdasaryan et al. 2020 BAGDASARYAN, E. et al. How to backdoor federated learning. In: PMLR. **International Conference on Artificial Intelligence and Statistics**. [S.l.], 2020. p. 2938–2948.

Baker et al. 2016 BAKER, B. et al. Designing neural network architectures using reinforcement learning. **arXiv preprint arXiv:1611.02167**, 2016.

Blazewicz, Lenstra e Kan 1983 BLAZEWICZ, J.; LENSTRA, J. K.; KAN, A. R. Scheduling subject to resource constraints: classification and complexity. **Discrete applied mathematics**, Elsevier, v. 5, n. 1, p. 11–24, 1983.

Cai, Zhu e Han 2018 CAI, H.; ZHU, L.; HAN, S. Proxylessnas: Direct neural architecture search on target task and hardware. **arXiv preprint arXiv:1812.00332**, 2018.

Cardoso et al. 2015 CARDOSO, M. C. et al. Quantum-inspired features and parameter optimization of spiking neural networks for a case study from atmospheric. **Procedia Computer Science**, Elsevier, v. 53, p. 74–81, 2015.

Chen et al. 2018 CHEN, Y. et al. Joint neural architecture search and quantization. **arXiv preprint arXiv:1811.09426**, 2018.

Chu, Zhang e Xu 2020 CHU, X.; ZHANG, B.; XU, R. Multi-objective reinforced evolution in mobile neural architecture search. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2020. p. 99–113.

Cruz 2007 CRUZ, A. Quantum-inspired evolutionary algorithms for problems based on numerical representation. **PhD Thesis**, Pontifical Catholic University of Rio de Janeiro, 2007.

Cruz, Vellasco e Pacheco 2007 CRUZ, A. A. da; VELLASCO, M. M. B. R.; PACHECO, M. A. C. Quantum-inspired evolutionary algorithm for numerical optimization. In: **Hybrid evolutionary algorithms**. [S.l.]: Springer, 2007. p. 19–37.

Cruz, Vellasco e Pacheco 2010 CRUZ, A. V. A. da; VELLASCO, M. M.; PACHECO, M. A. C. Quantum-inspired evolutionary algorithms applied to numerical optimization problems. In: IEEE. **IEEE Congress on Evolutionary Computation**. [S.l.], 2010. p. 1–6.

Deb 2014 DEB, K. Multi-objective optimization. In: **Search methodologies**. [S.l.]: Springer, 2014. p. 403–449.

Dong e Yang 2019 DONG, X.; YANG, Y. Searching for a robust neural architecture in four gpu hours. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2019. p. 1761–1770.

Eiben, Smith et al. 2003 EIBEN, A. E.; SMITH, J. E. et al. **Introduction to evolutionary computing**. [S.l.]: Springer, 2003. v. 53.

Elsken, Metzen e Hutter 2018 ELSKEN, T.; METZEN, J. H.; HUTTER, F. Efficient multi-objective neural architecture search via lamarckian evolution. **arXiv preprint arXiv:1804.09081**, 2018.

Fang et al. 2020 FANG, J. et al. Densely connected search space for more flexible neural architecture search. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. p. 10628–10637.

Geiping et al. 2020 GEIPING, J. et al. Inverting gradients–how easy is it to break privacy in federated learning? **arXiv preprint arXiv:2003.14053**, 2020.

Goldberg e Deb 1991 GOLDBERG, D. E.; DEB, K. A comparative analysis of selection schemes used in genetic algorithms. In: **Foundations of genetic algorithms**. [S.l.]: Elsevier, 1991. v. 1, p. 69–93.

Goodfellow et al. 2014 GOODFELLOW, I. et al. Generative adversarial nets. **Advances in neural information processing systems**, v. 27, 2014.

Gulli, Kapoor e Pal 2019 GULLI, A.; KAPOOR, A.; PAL, S. **Deep learning with TensorFlow 2 and Keras: regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API**. [S.l.]: Packt Publishing Ltd, 2019.

Han e Kim 2002 HAN, K.-H.; KIM, J.-H. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. **IEEE transactions on evolutionary computation**, IEEE, v. 6, n. 6, p. 580–593, 2002.

Han et al. 2020 HAN, S. et al. Optimal dnn architecture search using bayesian optimization hyperband for arrhythmia detection. In: IEEE. **2020 IEEE Wireless Power Transfer Conference (WPTC)**. [S.l.], 2020. p. 357–360.

He et al. 2015 HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: **Proceedings of the IEEE international conference on computer vision**. [S.l.: s.n.], 2015. p. 1026–1034.

He et al. 2016 HE, K. et al. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778.

He et al. 2016 HE, K. et al. Identity mappings in deep residual networks. In: SPRINGER. **European conference on computer vision**. [S.l.], 2016. p. 630–645.

He et al. 2020 HE, X. et al. Benchmarking deep learning models and automated model design for covid-19 detection with chest ct scans. **medRxiv**, Cold Spring Harbor Laboratory Press, 2020.

He et al. 2021 HE, X. et al. Efficient multi-objective evolutionary 3d neural architecture search for covid-19 detection with chest ct scans. **arXiv preprint arXiv:2101.10667**, 2021.

Hinton, Srivastava e Swersky 2012 HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. **Cited on**, v. 14, n. 8, p. 2, 2012.

Hitaj, Ateniese e Perez-Cruz 2017 HITAJ, B.; ATENIESE, G.; PEREZ-CRUZ, F. Deep models under the gan: information leakage from collaborative deep learning. In: **Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security**. [S.l.: s.n.], 2017. p. 603–618.

Hsu et al. 2018 HSU, C.-H. et al. Monas: Multi-objective neural architecture search using reinforcement learning. **arXiv preprint arXiv:1806.10332**, 2018.

Hutter, Kotthoff e Vanschoren 2019 HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated machine learning: methods, systems, challenges**. [S.l.]: Springer Nature, 2019.

Keskar e Socher 2017 KESKAR, N. S.; SOCHER, R. Improving generalization performance by switching from adam to sgd. **arXiv preprint arXiv:1712.07628**, 2017.

Laredo et al. 2019 LAREDO, D. et al. Automatic model selection for neural networks. **arXiv preprint arXiv:1905.06010**, 2019.

LeCun, Bengio e Hinton 2015 LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

Li e Talwalkar 2020 LI, L.; TALWALKAR, A. Random search and reproducibility for neural architecture search. In: PMLR. **Uncertainty in artificial intelligence**. [S.l.], 2020. p. 367–377.

Lin, Chen e Yan 2013 LIN, M.; CHEN, Q.; YAN, S. Network in network. **arXiv preprint arXiv:1312.4400**, 2013.

Liu, Simonyan e Yang 2018 LIU, H.; SIMONYAN, K.; YANG, Y. Darts: Differentiable architecture search. **arXiv preprint arXiv:1806.09055**, 2018.

Liu et al. 2021 LIU, Y. et al. A survey on evolutionary neural architecture search. **IEEE Transactions on Neural Networks and Learning Systems**, IEEE, 2021.

Loni et al. 2018 LONI, M. et al. Designing compact convolutional neural network for embedded stereo vision systems. In: IEEE. **2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)**. [S.l.], 2018. p. 244–251.

Lorraine, Vicol e Duvenaud 2020 LORRAINE, J.; VICOL, P.; DUVENAUD, D. Optimizing millions of hyperparameters by implicit differentiation. In: PMLR. **International Conference on Artificial Intelligence and Statistics**. [S.l.], 2020. p. 1540–1552.

Lu et al. 2019 LU, Z. et al. Multi-criterion evolutionary design of deep convolutional neural networks. **ArXiv**, abs/1912.01369, 2019.

Mittal et al. 2020 MITTAL, H. et al. Fake-face image classification using improved quantum-inspired evolutionary-based feature selection method. In: IEEE. **2020 IEEE Symposium Series on Computational Intelligence (SSCI)**. [S.l.], 2020. p. 989–995.

Mnih et al. 2015 MNIH, V. et al. Human-level control through deep reinforcement learning. **nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

Moore e Narayanan 1995 MOORE, M.; NARAYANAN, A. Quantum-inspired computing. **Dept. Comput. Sci., Univ. Exeter, Exeter, UK**, Citeseer, 1995.

Morozov et al. 2020 MOROZOV, S. et al. Mosmeddata: Chest ct scans with covid-19 related findings dataset. **arXiv preprint arXiv:2005.06465**, 2020.

Orr e Müller 1998 ORR, G. B.; MÜLLER, K.-R. **Neural networks: tricks of the trade**. [S.l.]: Springer, 1998.

Ottelander et al. 2021 OTTELANDER, T. D. et al. Local search is a remarkably strong baseline for neural architecture search. In: SPRINGER. **International Conference on Evolutionary Multi-Criterion Optimization**. [S.l.], 2021. p. 465–479.

Ozturk et al. 2020 OZTURK, T. et al. Automated detection of covid-19 cases using deep neural networks with x-ray images. **Computers in biology and medicine**, Elsevier, v. 121, p. 103792, 2020.

Pinho, Vellasco e Cruz 2009 PINHO, A. G. de; VELLASCO, M.; CRUZ, A. V. A. da. A new model for credit approval problems: A quantum-inspired neuro-evolutionary algorithm with binary-real representation. In: IEEE. **2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)**. [S.l.], 2009. p. 445–450.

Rahbar e Yazdani 2021 RAHBAR, M.; YAZDANI, S. Forecasting of covid-19 cases, using an evolutionary neural architecture search approach. **arXiv preprint arXiv:2109.13062**, 2021.

Rahimzadeh, Attar e Sakhaei 2021 RAHIMZADEH, M.; ATTAR, A.; SAKHAEI, S. M. A fully automated deep learning-based network for detecting covid-19 from a new and large lung ct scan dataset. **Biomedical Signal Processing and Control**, Elsevier, v. 68, p. 102588, 2021.

Ramos e Vellasco 2020 RAMOS, A. C.; VELLASCO, M. Chaotic quantum-inspired evolutionary algorithm: enhancing feature selection in bci. In: IEEE. **2020 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2020. p. 1–8.

Real et al. 2019 REAL, E. et al. Regularized evolution for image classifier architecture search. In: **Proceedings of the aaai conference on artificial intelligence**. [S.l.: s.n.], 2019. v. 33, n. 01, p. 4780–4789.

Real et al. 2017 REAL, E. et al. Large-scale evolution of image classifiers. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2017. p. 2902–2911.

Saad et al. 2021 SAAD, H. M. et al. Quantum-inspired genetic algorithm for resource-constrained project-scheduling. **IEEE Access**, IEEE, v. 9, p. 38488–38502, 2021.

Schumacher 1995 SCHUMACHER, B. Quantum coding. **Physical Review A**, APS, v. 51, n. 4, p. 2738, 1995.

Shin, Packer e Song 2018 SHIN, R.; PACKER, C.; SONG, D. X. Differentiable neural network architecture search. In: **ICLR**. [S.l.: s.n.], 2018.

Simonyan e Zisserman 2014 SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

Smith 2017 SMITH, L. N. Cyclical learning rates for training neural networks. In: IEEE. **2017 IEEE winter conference on applications of computer vision (WACV)**. [S.l.], 2017. p. 464–472.

Sun et al. 2019 SUN, S. et al. A survey of optimization methods from a machine learning perspective. **IEEE transactions on cybernetics**, IEEE, v. 50, n. 8, p. 3668–3681, 2019.

Sutton e Barto 2018 SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

Szegedy et al. 2015 SZEGEDY, C. et al. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9.

Szegedy et al. 2016 SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 2818–2826.

Szwarcman 2020 SZWARCMAN, D. **Quantum-inspired Neural Architecture Search**. Tese (Doutorado) — PUC-Rio, 2020.

Szwarcman, Civitarese e Vellasco 2019 SZWARCMAN, D.; CIVITARESE, D.; VELLASCO, M. Quantum-inspired neural architecture search. In: IEEE. **2019 International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 2019. p. 1–8.

Tan e Le 2019 TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2019. p. 6105–6114.

Tian et al. 2020 TIAN, Y. et al. Off-policy reinforcement learning for efficient and effective gan architecture search. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2020. p. 175–192.

Tieleman, Hinton et al. 2012 TIELEMAN, T.; HINTON, G. et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. **COURSERA: Neural networks for machine learning**, v. 4, n. 2, p. 26–31, 2012.

Timofeev, Chrysos e Cevher 2021 TIMOFEEV, A.; CHRYSOS, G. G.; CEVHER, V. Self-supervised neural architecture search for imbalanced datasets. **arXiv preprint arXiv:2109.08580**, 2021.

Wan et al. 2020 WAN, A. et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. p. 12965–12974.

Wang, Lin e Wong 2020 WANG, L.; LIN, Z. Q.; WONG, A. Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. **Scientific Reports**, Nature Publishing Group, v. 10, n. 1, p. 1–12, 2020.

Wang et al. 2017 WANG, X. et al. Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In: **IEEE CVPR**. [S.l.: s.n.], 2017. v. 7.

Wang et al. 2019 WANG, Z. et al. Beyond inferring class representatives: User-level privacy leakage from federated learning. In: IEEE. **IEEE INFOCOM 2019-IEEE Conference on Computer Communications**. [S.l.], 2019. p. 2512–2520.

Watkins 1989 WATKINS, C. J. C. H. Learning from delayed rewards. King's College, Cambridge United Kingdom, 1989.

Wilson et al. 2017 WILSON, A. C. et al. The marginal value of adaptive gradient methods in machine learning. **Advances in neural information processing systems**, v. 30, 2017.

Wistuba, Rawat e Pedapati 2019 WISTUBA, M.; RAWAT, A.; PEDAPATI, T. A survey on neural architecture search. **arXiv preprint arXiv:1905.01392**, 2019.

Xie et al. 2018 XIE, S. et al. Snas: stochastic neural architecture search. **arXiv preprint arXiv:1812.09926**, 2018.

Yang 2010 YANG, X.-S. A new metaheuristic bat-inspired algorithm. In: **Nature inspired cooperative strategies for optimization (NICSO 2010)**. [S.l.]: Springer, 2010. p. 65–74.

Yang et al. 2020 YANG, Z. et al. Cars: Continuous evolution for efficient neural architecture search. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. p. 1829–1838.

Ye et al. 2020 YE, W. et al. Quantum-inspired evolutionary algorithm for convolutional neural networks architecture search. In: IEEE. **2020 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2020. p. 1–8.

Zhang 2019 ZHANG, L. M. A new compensatory genetic algorithm-based method for effective compressed multi-function convolutional neural network model selection with multi-objective optimization. **arXiv preprint arXiv:1906.11912**, 2019.

Zhong et al. 2018 ZHONG, Z. et al. Practical block-wise neural network architecture generation. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2018. p. 2423–2432.

Zhu, Zhang e Jin 2021 ZHU, H.; ZHANG, H.; JIN, Y. From federated learning to federated neural architecture search: a survey. **Complex & Intelligent Systems**, Springer, v. 7, n. 2, p. 639–657, 2021.

Zoph e Le 2016 ZOPH, B.; LE, Q. V. Neural architecture search with reinforcement learning. **arXiv preprint arXiv:1611.01578**, 2016.

Zoph et al. 2018 ZOPH, B. et al. Learning transferable architectures for scalable image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2018. p. 8697–8710.

A
Implementation Details

One of the main modifications of Q-NAS was to update the Tensorflow version from 1.9 to 2.3. The Tensorflow is an open source deep learning library and it was used to make the network assembling and training. There are huge changes between these versions (Gulli, Kapoor e Pal 2019). Using MPI4Py library, Q-NAS was implemented to run in a multi-process environment via MPI messages.

The master node runs Q-NAS and distributes the evaluation tasks to the slaves with non-blocking send operations and it also collects the results with non-blocking receives. The master and the slaves evaluates one individual per generation. This means that the number of processes is equal to the number of classical individuals to be evaluated.

The experiments were executed in a multi-computer environment, which contains 4 NVIDIA GTX 1080 running on Linux system. OpenMPI 3.1.1 distribution was installed. However, we run the jobs on a single machine from the environment described above for the retraining phase. We use 1 GPU and 1 CPU in the single machine.