Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

July 4, 2022

Q-NAS Applied to the Classification of Medical Images

Giovanna Avelar Espozel

PUC RIO

Projeto de Graduação

DEPARTAMENTO
DE ENGENHARIA
ELÉTRICA

Q-NAS Applied to the Classification of Medical Images

Student: Giovanna Avelar Espozel

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

Work presented as partial requirement to the conclusion of the Electrical Engineering major in the Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.

Acknowledgements

First, I would like to express my sincere gratitude to my advisor Professor Marley Vellasco and co-adivisor Professor Karla Figueiredo for their insightful comments and guidance that have helped me immensely during this project.

I would also like to thank PUC-Rio for all of the opportunities that I was able to have during my years as a student. I am grateful for all of the faculty who contributed to my learning.

I wish to thank my mom, Ana Avelar, who supported me in all of my decisions and was always there to offer advice when I needed. Without her encouragement, my academic trajectory would have been impossible.

I would like to thank my brother, Pedro Espozel, for his company in all of my academic years. With him beside me, I knew I would always have a friend to face new experiences with.

To my late father, Alexandre Espozel, who taught me to always dream big and to never be afraid of chasing what you want.

I would also like to thank Ted and Nick who never failed to make me laugh.

To the Reptiles Team and all of its members with whom I had great adventures and new experiences.

To my university friends who complemented these past five years and are the reason I had a great college experience.

Finally, I wish to thank to everyone who contributed to my graduation.

Abstract

This Course Conclusion Project consists in classifying images obtained from chest computed tomography scans from patients who have had COVID-19 before by using Artificial Intelligence. The Q-NAS model will be applied to these images and will classify them in six different classes according to the post-covid lung pattern found. The purpose of this undergraduate project is to find a neural network capable of classifying the post-covid patterns with precision and speed.

Keywords: Convolutional Neural Networks, Post-Covid Patterns, Neural Architecture Search, Medical Images Classification

Q-NAS Aplicado à Classificação de Imagens Médicas

Resumo

Este trabalho de conclusão de curso consiste em classificar imagens obtidas de tomografias computadorizadas do tórax de pacientes que já tiveram COVID-19 através de Inteligência Artificial. Pretende-se classificar essas imagens com o auxílio do algoritmo Q-NAS em seis diferentes classes de acordo com o padrão pulmonar pós-covid identificado. O propósito deste projeto de graduação é encontrar uma rede neural que possa realizar a classificação dos padrões pós-covid com precisão e rapidez.

Palavras-chave: Redes Neurais Convolucionais, Padrões Pós-Covid, Busca de Arquiteturas Neurais, Classificação de Imagens Médicas

Sumário

List of Figures

List of Tables

1 Introduction

a Motivation

The "Severe Acute Respiratory Syndrome Coronavirus 2" (SARS-CoV-2), commonly known as COVID-19, first appeared in Wuhan, China, in December of 2019 [4]. By March of 2020, it had already spread to different countries and the World Health Organization (WHO) declared the COVID-19 a pandemic [5]. Since then, most countries have enforced lockdowns for different periods of time in order to contain the spread of the virus that has had millions of confirmed cases and caused millions of deaths worldwide and hundreds of thousands of deaths in Brazil alone [6].

Due to its novelty, we still do not know all the consequences that the virus can cause and whether they are permanent or not. Therefore, in a lot of different countries, studies are being developed in order to better understand the disease.

In this study, we aim to use artificial intelligence in order to analyse computed tomography chest images taken from patients from Pedro Ernesto University Hospital who had COVID-19 before. By doing this, we hope to aid medical practitioners in identifying the effects that COVID-19 may have on our lungs.

b Objectives

This work aims to identify different types of patterns that can appear on someone's lungs after they have had COVID-19. This was done by applying the Quantum-Inspired Neural Architecture Search (Q-NAS) model [3] to the chest computed tomography (CT) images dataset. The aim of the neural networks developed by this model is to classify the patterns present on the lungs' images in six different categories: Normal, Fibrosis, Nonspecific Interstitial Pneumonia (NSIP or PINE in Portuguese), Reabsorption, Airway, and Other. These classes were chosen according to the analysis done in [7].

The Q-NAS model is written in the Python programming language which has several modules made for deep learning applications. This model was chosen due to its ability to create multiple neural networks, evaluate their performance and automatically choose the network with the best classification accuracy when applied to any dataset, which saves the user a lot of time when designing their own network as they won't need to manually decide the many possible configurations of a neural network.

c Organization

This work has four additional chapters, which are described bellow.

In Chapter 2 there is the explanation of the key concepts necessary in order to understand how does the Q-NAS model work. This includes a brief introduction to deep learning, convoluted networks, neural architecture search and quantum-inspired evolutionary algorithms.

Chapter 3 describes the patterns that we want to identify in the images and their impacts on the lungs.

In Chapter 4, we explain how the dataset was manipulated in order to be used by the Q-NAS model and which configurations were used.

Chapter 5 presents the results obtained from different experiments.

Finally, the conclusions and final remarks for this work are presented in Chapter 6.

2 Quantum Inspired Neural Architecture Search (Q-NAS)

In order to understand what is the Q-NAS and how does it work, a few key concepts will be briefly explained in the following sections.

a Deep Learning

”AI is a field that includes a broad set of approaches with the goal of creating machines with intelligence. Deep learning is only one such approach. Deep learning is itself one method among many in the field of machine learning, a subfield of AI in which machines “learn” from data or from their own “experiences.”” [8]

”A deep learning algorithm is a particular kind of representation learning procedure that discovers multiple levels of representation, with higher-level features representing more abstract aspects of the data.” [9]

Over the years, lots of different deep learning algorithms were proposed to solve real world problems, such as classifications or predictions involving a lot of data. One of these algorithms is the Convolutional Neural Networks (CNN’s) which work well with image classification problems. These networks were used in the Q-NAS and will be explained in the following section.

b Convolutional Neural Networks

Convolutional Neural Networks (CNN’s) [10] are a type of artificial neural networks widely used in image classification problems. Their architecture and organization allow them to extract features from images, capture spatial dependencies using kernels (filters), and recognize visual patterns.

Their layer structure can be divided in different types of functions that can be repeated many times throughout the network. The first layer receives the input, which in our case is an image. In this case, the input is usually represented by a three-dimensional matrix where the size of the matrix (width and height) is determined by the image dimensions (pixels) and the depth is determined by the number of colour channels. For example, in an image represented by RGB, there are three colour channels, but other colour models, such as Greyscale and CMYK, can be used.

We can imagine that the computational effort would be very high if the images are too big. Therefore, reducing these images to a form that makes it easier to process them would be better. The CNN’s can do that, although we need to be careful when reducing the image to avoid losing any feature that might be essential to the classification. In the next sections, the key layers and functions are described.

1 Convolutional Layer

We can deduce from the name that this type of network has at least one layer that uses the mathematical operation called convolution. In the equation bellow, we can see the convolution of a function *x(t)* with a function *w(t)*.

$$s(t) = (x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)\,da \tag{1}$$

In machine learning, the convolution operation described above is written in the following format as described by Goodfellow et al. [11]:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2}$$

In the equation above, *x* is now, *I* which represent the input, *w* is called *K* for kernel (filter), the output *s* is now the feature map, *m* and *n* are the pixels coordinates in the local receptive field (group of pixels where the convolution is being applied at the moment) and *i* and *j* are the coordinates of each pixel. Both *I* and *K* are multidimensional arrays where *I* is a data array and *K* is a parameters array which the learning algorithm can update. In the Figure 1 we can see a visual example of a 2D convolution.

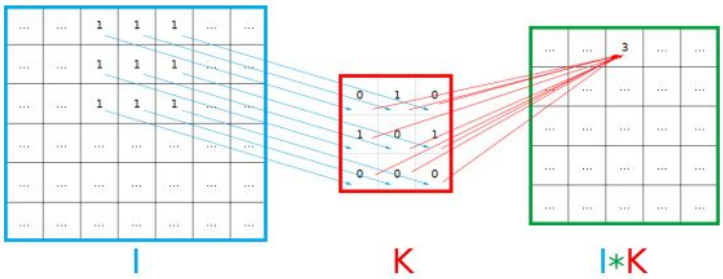

Figure 1: Example of convolution in a local receptive field, where, compared to Equation (5) *m* = 3,4,5 and *n* = 1,2,3 [1]

We can conclude from Figure 1, that by using the kernels to scan the whole image, the filter's parameters will be shared between the pixels, reducing their quantity and making the network more efficient. We can also see that if the the kernels' sizes are smaller than the image's size (which is generally the case), not all of the input pixels will influence the output in the feature map, this reduces the amount of operations and connections in the network, and it also improves it's efficiency [11]. Therefore, by using these filters, we can identify local patterns in the image.

By making the kernel smaller than the image, we notice that the feature map will also be smaller than it's input. This can be controlled by adding *padding* (pixels with value zero) around the input or we can adjust the kernel step (*stride*) used to scan the image.

As it is possible to see, there are many decisions to be made when creating a convolutional layer and most of them are related to the kernel: we need to know how many kernels will be used, their size and stride. All of these decisions influence a lot the layer's performance.

2 Activation Function

Activation functions are used after a convolutional layer to bring non-linearity to the networks, which allows them to learn more complex tasks. These functions can be of different types such as Sigmoid, Hyperbolic Tangent and ReLu (Rectified Linear Unit), which are described respectively in the equations bellow.

$$\sigma(x) = (1 + e^{-x})^{-1} \tag{3}$$

$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4}$$

$$\sigma(x) = max(0, x) \tag{5}$$

Of all activation functions, the ReLu is the one generally used with convolutional networks for image classification. By analysing equations (3),(4) and (5), we observe that the ReLu does not use time consuming operations like sigmoid and hyperbolic tangent do, and that is why the networks that use ReLu tend to be faster than the ones that use those other functions, for example [12].

3 Pooling Layer

Besides the convolutions layers, CNN's use pooling layers, since they reduce the size of the feature maps. Consequently, they reduce the number of parameters and connections needed for future layers in the network. According to Chollet [13], by reducing the size of feature maps before a block of densely connected layers, we can reduce overfitting.

The pooling layers scan the input in rectangular slices of size *k x k*, where k is usually 2 or 3 [1], and they return the maximum or medium values for each slice. In order to return these values, the function used in the pooling layer can be Max-Pooling or Average-Pooling, respectively. In Figure 2 we can see how these functions work. Therefore, the idea behind the pooling layers is that after identifying a pattern, its exact location is not as important as its approximate location in relation to other patterns [11].

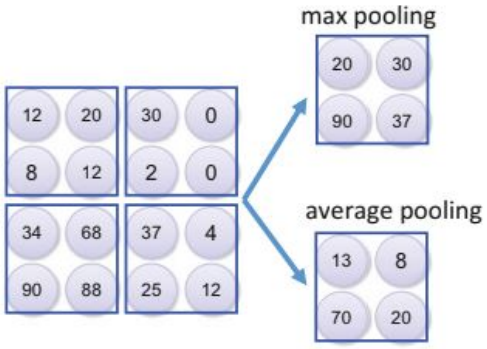

Figure 2: Representation of the functions Max-pooling and Average-Pooling [2]

A CNN is usually built by stacking different pooling and convolutional layers. The final layers are usually fully connected, which means neurons from one layer are connected to neurons on the following layer, as typically seen in conventional neural networks such as Multilayer Perceptrons. Because of these connections, when these layers receive the feature maps from the previous layer, they are able to identify which is the correct class for the input image. Therefore, these layers are the ones responsible for doing the classification.

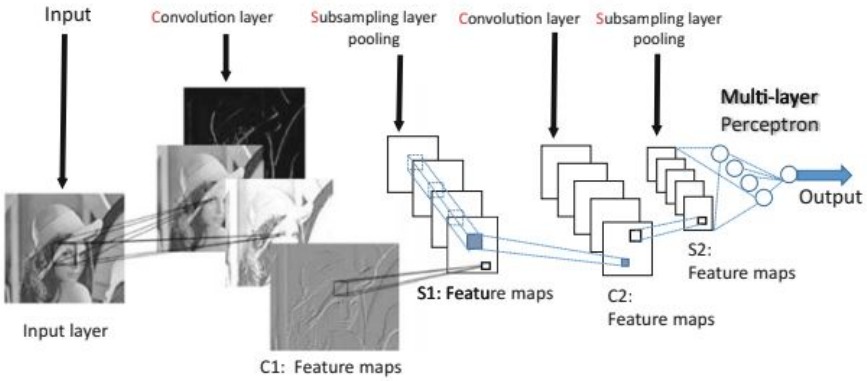

Figure 3: Example of a convolutional neural network [2]

4 Training and Generalization

The objective of the networks that we want to create is to classify images, which means that we want to make sure that these networks have minimum errors when classifying images that they have not seen before in a process called generalization. In order to evaluate if a network is making a lot of mistakes, we use error functions (cost functions) to analyse its progress.

Since we want to have less errors when classifying a dataset, we can use different techniques to minimize the error function during training. One of them is using an optimizer, an algorithm that minimizes the training error by altering the network's weights. Some types of optimizers are RMSProp (Root Mean Squared Propagation) which is an adaptive learning rate method [14] and SWATS (Switching from ADAM to SGD) [15] which is an strategy that uses two other optimizers, SGD (Stochastic Gradient Descent) [16] and Adam (Adaptive Moment Estimation) [17].

In addition to the optimizer, other strategies can be used, such as the weight decay technique, which allows the network to penalize the cost function [11] and the dataset augmentation technique, increasing the dataset. It increases the training set by adding fake data to it, which can be done by flipping the original images, changing their brightness or contrast and other similar alterations [11]. This is usually done when there is limited amount of data available.

Other strategies which are batch normalization [18] and the use of residual functions [19]. By implementing batch normalization, we can reestablish the normalization of the parameters after each training mini-batch, this makes the normalization a part of the model architecture instead of only being used in

the initial parameters and enables a higher learning rate, reduces training time, allows the user to be less careful with the parameter initialization and prevents overfitting [18] [20] [21]. In fact, overfitting can also be prevented by using a strategy called early stopping, in which the training is interrupted when no improvement is detected in the network's performance.

The idea behind introducing residual functions to the network is to gain accuracy from the increased depth and make the networks easier to optimize [19]. The residual learning framework introduces *skip-connections* (shortcuts) between convolutional blocks. These shortcuts are identity functions and residual functions learned by the network. The *skip-connections* make it easier for the residual network to identify which layers do not contribute to the overall performance and they allow the network to skip these layers by using the identity functions. In the Q-NAS model, the user can choose to evolve a network by using these residual functions or choose to use only convolutional (with an activation function) and pooling layers. The user can also decide which combination of the techniques mentioned before will be used, creating numerous configurations [3] [22].

c Neural Architecture Search (NAS)

As we can see, there are a lot of decisions to be made when creating a CNN, they can be regarding the architectures which implies deciding the quantity of layers, the configuration of filter to be used, for example, but they can also be regarding the values of the hyperparameters such as the learning rate, the optimizer settings and many others. All of these decisions greatly influence the performance of the network, if it is fast or has a good accuracy. Because of all these possibilities when designing a network, several algorithms with different approaches were created over the years to automatize and facilitate this process in a neural architecture search technique. The Quantum Inspired Neural Architecture Search model [3] is one approach that is based on quantum-inspired evolutionary algorithms. The following sections will briefly describe how this type of algorithm works.

1 Quantum Inspired Evolutionary Algorithm

The idea behind evolutionary algorithms is that we want to optimize a fitness function. In order to do that, various solutions are proposed, which are called individuals and together they are a population. These individuals are then evaluated by applying the quality function receiving a fitness value. Since the process of evolution inspires the evolutionary algorithms, they also respect the survival of the fittest. Therefore, the fittest individuals of each generation (iteration of the algorithm) are selected to form a new population. Some operations used to create a new population are recombination, mutation or crossover [23] [3].

The quantum-inspired evolutionary algorithm applies quantum computing principles to enhance classical evolutionary algorithms [24]. In the algorithm proposed by Han and Kim [25] some of the quantum principles applied were the quantum-bit or q-bit and superposition of states. Analogue to a classical computer, the smallest unit of information stored in a two-state quantum computer is the q-bit [24]. The q-bit state in state "1", "0" or in a superposition of both states; therefore, the state can be described by the following equation [25] [24]:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{6}$$

Where $|\Psi\rangle$ is the q-bit state, α and β are complex numbers and $|\alpha|^2$ and $|\beta|^2$ gives the probability that the q-bit will be found in the "0" state and "1" state respectively [25].

Therefore, in the QIEA we now have populations of q-bit individuals where the quantum individual represents initially all possible states with the same probability which better characterizes the population diversity. These populations are called quantum populations, and they are the fundamental concept that differentiates classical evolutionary algorithms from the quantum-inspired ones.They represent a superposition of states, covering the search space, or more specifically, each quantum state characterizes a possible solution [3].

2 Q-NAS

The Q-NAS model, as mentioned before, is a quantum-inspired evolutionary algorithm. It automatically designs and builds deep neural networks to execute a predefined task [3].

In Q-NAS the network structure design and the hyperparameters optimization are addressed. This is done by dividing the quantum chromosome contained in the quantum individuals in two parts: one responsible for encoding the architecture space and the other for encoding the numerical space of some hyperparameters. More details about how the Q-NAS does the codification and the automatic generation of CNN

architectures can be found in [3]. In Figure 4 we can see the Q-NAS flowchart where the dark boxes highlight steps where each part of the chromosomes is executed separately.

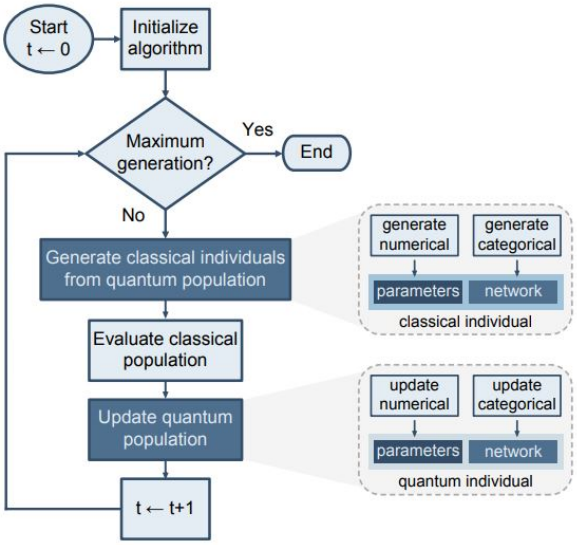

Figure 4: Q-NAS flowchart [3]

In Figure 5 we can see the summarized steps of the Q-NAS model, where $Q(t)$ is the quantum population, $C(t)$ is the classical population and $P(t)$ is the saved classical population.

1: $t \leftarrow 0$
2: Initialize $Q(t)$
3: **while** $t \leq T$ **do**
4: Generate classical population $C(t)$ observing $Q(t)$
5: **if** $t = 0$ **then**
6: Evaluate $C(t)$
7: $P(t) \leftarrow C(t)$
8: **else**
9: $C(t) \leftarrow$ recombination between $C(t)$ and $P(t)$
10: Evaluate $C(t)$
11: $P(t) \leftarrow$ best individuals from $[C(t) \cup P(t)]$
12: **end if**
13: $Q(t+1) \leftarrow$ update $Q(t)$ based on $P(t)$ values
14: $t \leftarrow t+1$
15: **end while**

Figure 5: Q-NAS flowchart [3]

3 Post Covid Patterns

In the chest CT images used in [7] different patterns were identified in patients from Pedro Ernesto University Hospital who have had COVID-19. These patterns were identified by analysing the numerous images generated in each chest CT. These images could be up to 400 depending on the patient and each was labeled in a different class [7]. These patterns and their classes (Normal, Fibrosis, NSIP, Reabsorption, Airway, and Other) were used in the Q-NAS model and they will be briefly described in the following sections.

a Normal

This class corresponds to the images in which no post-covid pattern were identified in the lungs of the patients and in the dataset there were 500 images available. An example of a chest computed tomography image for this pattern is in Figure 6:

Figure 6: Chest Computed Tomography Image with ”Normal” pattern

b Fibrosis

This class is for the images in which fibrosis-like lesions were found. Our dataset had 312 images for this class and an example can be seen in Figure 7.

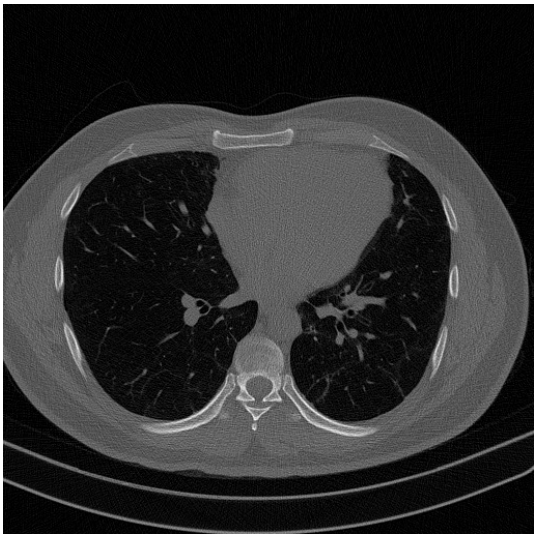

Figure 7: Computed Tomography Chest Image with ”Fibrosis” pattern

c Nonspecific Interstitial Pneumonia (NSIP or PINE in Portuguese)

The images in this class were the ones in which the pattern for lesions similar to nonspecific interstitial pneumonia (NSIP) was found. This class had 447 images in the dataset and the pattern can be observed in Figure 8.

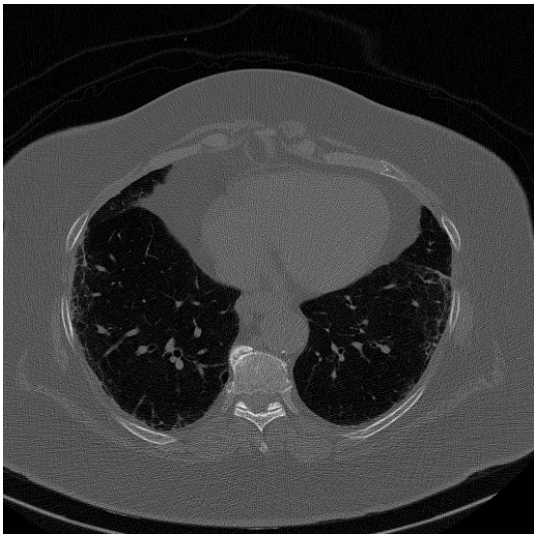

Figure 8: Computed Tomography Chest Image with ”NSIP” pattern

d Reabsorption

This class is for the images in which the reabsorption tomographic pattern was identified. According to [7] this pattern was the most common one in the patients with a post-covid pattern and in our dataset, 494 images belonged to this class. Figure 9 is an example of this pattern.

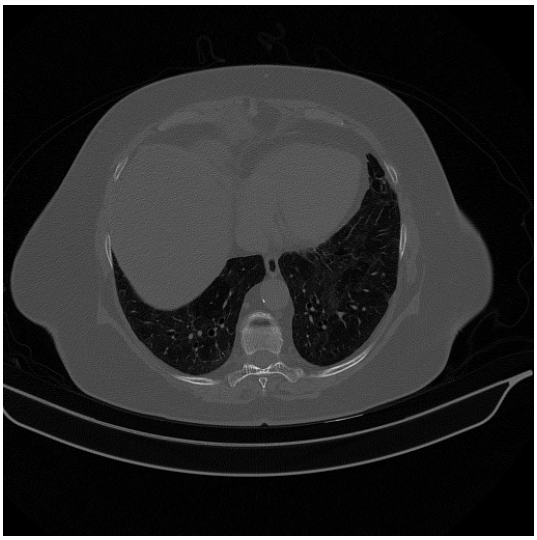

Figure 9: Computed Tomography Chest Image with ”Reabsorption” pattern

e Airway

This is the last pos-covid class and it belongs to the images in which airway disease patterns were found. This was the second most common post-covid pattern found in patients [7]. In our dataset we had 200 images in this class and in Figure 10 we can see an example of this pattern.

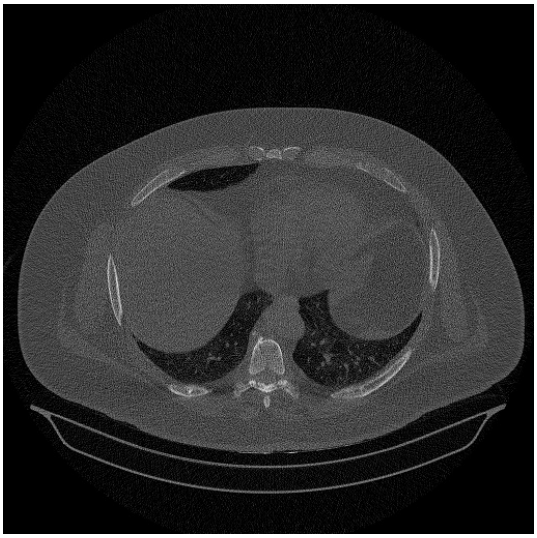

Figure 10: Computed Tomography Chest Image with ”Airway” pattern

f Other

As it was mentioned on the beginning of Section 3, each chest CT generates several images. These images are not necessarily of the lung, as the machine scans the chest they may produce some images of bones or other organs for example. Therefore, this class includes all of the images which could not be identified as one of the previously described classes which is why in the dataset we had 10074 images for this class. Figure 11 shows a type of image which belongs to this class.

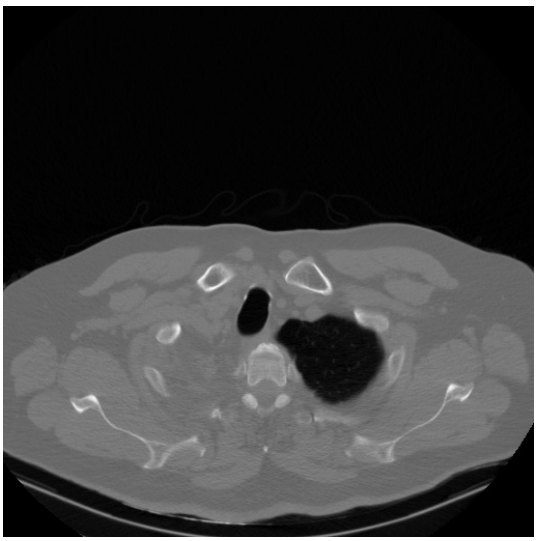

Figure 11: Computed Tomography Chest Image with ”Other” pattern

4 Results

As mentioned in Section 2, the Q-NAS model can evolve the network architecture as well as the hyperparameters. Since the experiments done in [3] with fixed hyperparemeters were the ones with better performances, the hyperparameters were also fixed in this work. Their values were taken from [3] as they provided good results. Some of the evolution parameters were also taken from [3] and remained the same for all of the experiments. Table 1 shows the hyperparameters and evolution parameters configuration shared among all of the experiments.

Table 1: Parameter and Hyperparameter configuration of the Q-NAS model

Parameter	Value
crossover_rate	0.5
num_quantum_ind	2
repetition	2
update_quantum_gen	5
update_quantum_rate	0.1
penalize_number	8
max_num_nodes	20
decay	0.9
learning_rate	1.0e-3
momentum	0.9
weight decay	1.0e-4

The parameter *max_num_nodes* refers to the network size, while the *num_quantum_ind* and *repetition* are the number of quantum individuals in the quantum population and the number of classical individuals each quantum individual will generate respectively. The parameter *update_quantum_gen* is the generation frequency to update quantum genes and *update_quantum_rate* is rate for all the quantum update operations. We also have *penalize_number* which is the maximum number of reducing layers a network can have without suffering penalization.

The number of structures produced per generation was set to 4 due to the limited number of GPU's available. After, deciding the configuration, the dataset was prepared in order to be applied to the Q-NAS. Since there was a limitation of images available and some of the classes had more images than others, in order to have a balanced dataset, 160 images were randomly chosen from each class to be used for training, while 40 images from each class would be for the validation set. The number of 160 images per category was determined by the class with the least amount the images. In this dataset, there were also 44 images per class for the test set. After the images were chosen, they were resized to 256x256 and converted to the format accepted by the Q-NAS.

The Q-NAS model has a setting wich applies data augmentation in the training set only. This augmentation step includes the addition of zero padding to the image as well as randomly cropping and flipping them.The first tests done were with the configuration used to find the best network from [22]. Therefore, we had *max_generations*=100 and *max_epochs*=50. The retrain scheme chosen for this experiment was the cosine scheme based on the results obtained from [3]. The function set used is specified in Table 2.

This setup had three runs, and the number of layers, retrain validation and retrain test accuracy and fitness for each one of the runs was collected. The results of these runs are listed in Table 3.

Table 2: Layer functions

function name	function	kernel size	stride	filters	initial probability
conv_1_1_32	ConvBlock	1	1	32	0.042
conv_1_1_64	ConvBlock	1	1	64	0.042
conv_3_1_32	ConvBlock	3	1	32	0.042
conv_3_1_64	ConvBlock	3	1	64	0.042
conv_3_1_128	ConvBlock	3	1	128	0.042
conv_3_1_256	ConvBlock	3	1	256	0.042
conv_5_1_32	ConvBlock	5	1	32	0.042
conv_5_1_64	ConvBlock	5	1	64	0.042
max_pool_2_2	MaxPool	2	2	–	0.167
avg_pool_2_2	AvgPool	2	2	–	0.167
no_op	NoOp	–	–	–	0.333

Table 3: Results for the augmentation experiment

#	# of layers	# parameters	accuracy		
			fitness	retrain validation	retrain test
1	16	0.48M	0.92500	0.9250	**0.89394**
2	12	0.36M	0.91250	0.90833	0.89394
3	15	0.48M	0.91667	0.83750	0.81818

As we can see, this configuration had some great results, and produced a network with a validation accuracy of 0.925% and most importantly, a test accuracy of 89.394%. The network's architecture is in Figure 12 and the confusion matrix for the test's results is in Figure 13.

Node	Funtion Name
1	avg_pool_2_2
2	no_op
3	conv_3_1_32
4	avg_pool_2_2
5	avg_pool_2_2
6	conv_3_1_32
7	no_op
8	no_op
9	conv_3_1_32
10	no_op
11	avg_pool_2_2
12	conv_3_1_32
13	conv_5_1_32
14	conv_3_1_256
15	max_pool_2_2
16	conv_5_1_32
17	conv_5_1_64
18	conv_1_1_64
19	max_pool_2_2
20	conv_3_1_128

Figure 12: Network Architecture for network #1

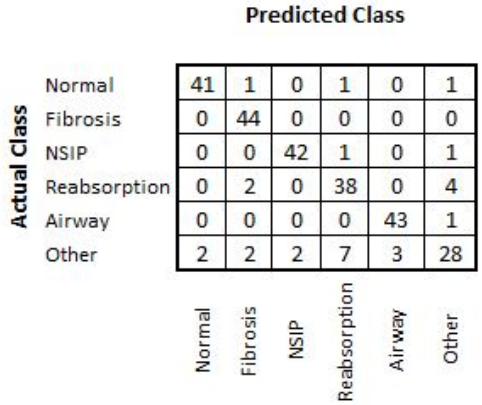

Figure 13: Confusion Matrix for network #1

The confusion matrix shows that the network classifies most of the classes without any trouble. The problem lies in the Other class. As it was mentioned in Section 3, this class contains any image that could not be classified in any other class. This means that these images have different patterns put together in the same class, which confuses the network.

In order to improve the test accuracy, in the next experiment the function set was expanded to the one specified in Table 4. In this new function set we had more convolutional functions which included different filter options.The other parameters were kept the same as the ones used in the first experiment in Table 3. The results for the new function set are listed in Table 5.

Table 4: Expanded function set

function name	function	kernel size	stride	filters	initial probability
conv_1_1_64	ConvBlock	1	1	64	0.028
conv_1_1_128	ConvBlock	1	1	128	0.028
conv_1_1_256	ConvBlock	1	1	256	0.028
conv_1_1_512	ConvBlock	1	1	512	0.028
conv_3_1_64	ConvBlock	3	1	64	0.028
conv_3_1_128	ConvBlock	3	1	128	0.028
conv_3_1_256	ConvBlock	3	1	256	0.028
conv_3_1_512	ConvBlock	3	1	512	0.028
conv_5_1_64	ConvBlock	5	1	64	0.028
conv_5_1_128	ConvBlock	5	1	128	0.028
conv_5_1_256	ConvBlock	5	1	256	0.028
conv_5_1_512	ConvBlock	5	1	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 5: Results for the expanded function set

#	# of layers	# parameters	accuracy		
			fitness	retrain validation	retrain test
1	12	0.56M	0.91667	0.92917	0.88258
2	13	3.65M	0.92083	0.92500	**0.90909**
3	15	3.86M	0.92083	0.9125	0.86742

Table 5 shows that the Q-NAS found a network with a test accuracy of 90.909% which is better that the one from Table 3. Network #2, has less layers that network #1 from experiment 1, and a validation

accuracy of 92.500%. Figures 14 and 15 show the architecture and confusion matrix for network #2 from Table 5.

Node	Funtion Name
1	max_pool_2_2
2	no_op
3	max_pool_2_2
4	no_op
5	max_pool_2_2
6	conv_3_1_64
7	avg_pool_2_2
8	conv_1_1_128
9	avg_pool_2_2
10	avg_pool_2_2
11	no_op
12	conv_1_1_512
13	conv_5_1_256
14	max_pool_2_2
15	conv_3_1_128
16	no_op
17	no_op
18	avg_pool_2_2
19	no_op
20	no_op

Figure 14: Network Architecture for network #2

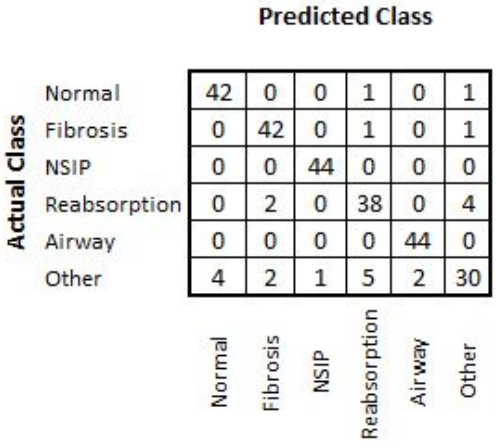

Figure 15: Confusion Matrix for network #2

In Figure 15 we see that network #2 also has the most trouble with the class Other as expected. In the next experiment, the parameters *max_epochs* and *max_generations* were altered. This test was to check if the Q-NAS could find a better network or one as good as #2 from Table 5 with less generations and epochs. Therefore, the new values for these parameters were *max_epochs*=30 and *max_generations*=70. The results of this experiment are listed in Table 6.

Table 6: Results for the experiment with less epochs and generations

#	# of layers	# parameters	accuracy		
			fitness	retrain validation	retrain test
1	12	2.83M	0.90417	0.93333	0.89015
2	14	3.97M	0.91250	0.90000	0.90530
3	14	0.42M	0.87500	0.92500	**0.90909**

As we can see, network #3 has the test accuracy of 90.909% and validation accuracy of 90.2500% which are the same ones found in the previous experiment. This means that with this dataset the Q-NAS can find good networks within 70 generations and 30 epochs only. In Figure 16 we can see the architecture evolved by Q-NAS for this network.

Node	Funtion Name
1	max_pool_2_2
2	avg_pool_2_2
3	conv_5_1_64
4	conv_3_1_64
5	max_pool_2_2
6	no_op
7	conv_1_1_256
8	max_pool_2_2
9	no_op
10	no_op
11	avg_pool_2_2
12	conv_1_1_256
13	max_pool_2_2
14	conv_3_1_64
15	max_pool_2_2
16	avg_pool_2_2
17	no_op
18	conv_3_1_256
19	no_op
20	no_op

Figure 16: Network Architecture for network #3

And in Figure 17 we have the confusion matrix. As we expected from previous tests, the worst class continues to be Other.

Predicted Class

Actual Class	Normal	Fibrosis	NSIP	Reabsorption	Airway	Other
Normal	43	0	0	1	0	0
Fibrosis	0	44	0	0	0	0
NSIP	0	0	43	0	0	1
Reabsorption	0	2	0	37	0	5
Airway	0	0	0	0	43	1
Other	2	2	1	4	5	30

Figure 17: Confusion Matrix for network #3

a New Datasets

In order to see if the accuracy would increase without the class Other, two new datasets with only 5 classes were created. In both of them the classes Fibrosis, NSIP, Reabsorption and Airway were kept the same, the difference is that in the first dataset we combined the classes Normal with Other following the suggestion from [7] and the second dataset was created by excluding the class Other, while all of the other classes remained the same.

1 Dataset with classes Normal and Other combined

In order to prepare this dataset, 80 random images were selected from the Normal and Other classes to compose the training images for the new class. For the validation and test sets, we took 20 and 22 images, respectively, from each class. The configuration used in the Q-NAS is the same for the one used for Table 6. The results for this experiment are in Table 7.

Table 7: Results for the experiment with less epochs and generations

#	# of layers	# parameters	accuracy		
			fitness	retrain validation	retrain test
1	14	1.11M	0.92000	0.93500	**0.94091**
2	11	1.83M	0.95500	0.95500	0.92273
3	12	1.97M	0.95500	0.95500	0.92727

All of the networks found in Table 7 have a better test accuracy than the ones in Tables 5 and 6, as desired. As we can see, the best test accuracy is 94.091% with the validation accuracy of 93.500%. In Figures 18 and 19 we can see the architecture and confusion matrix for this network, respectively.

Node	Funtion Name
1	max_pool_2_2
2	no_op
3	max_pool_2_2
4	no_op
5	max_pool_2_2
6	no_op
7	conv_1_1_128
8	conv_3_1_256
9	max_pool_2_2
10	conv_3_1_128
11	max_pool_2_2
12	no_op
13	conv_5_1_64
14	no_op
15	conv_3_1_256
16	no_op
17	conv_3_1_64
18	avg_pool_2_2
19	max_pool_2_2
20	conv_1_1_256

Figure 18: Network Architecture for network #1

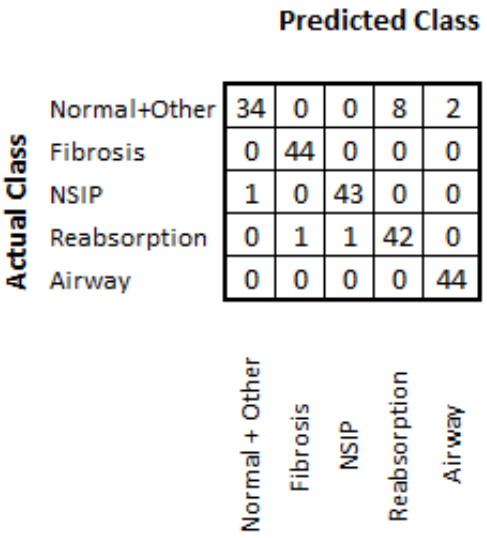

Figure 19: Confusion Matrix for network #1

2 Dataset without class Other

In this dataset all of the classes had 160 images as before, and the only preparation done was taking out the class Other. The Q-NAS configuration was kept the same as the one for Table 7 and the results for this experiment are listed in Table 8.

Table 8: Results for the experiment with less epochs and generations

#	# of layers	# parameters	accuracy		
			fitness	retrain validation	retrain test
1	14	1.26M	0.97000	0.99000	0.98636
2	14	1.07M	0.97500	0.99500	**0.98636**
3	13	2.76M	0.99000	0.98000	0.98182

As expected the results found with this dataset are much better than the ones found in all of the other experiments as we don't have the class Other which was the reason for the majority of the errors in the previous tests. In Table 8 we can see that the best network has a test accuracy of 98.636% and a validation accuracy of 99.500%, and in Figure 20 we can see the network architecture while, in Figure 21 we the confusion matrix.

Node	Funtion Name
1	max_pool_2_2
2	no_op
3	no_op
4	no_op
5	no_op
6	max_pool_2_2
7	no_op
8	no_op
9	conv_3_1_128
10	no_op
11	conv_5_1_64
12	no_op
13	no_op
14	conv_3_1_64
15	max_pool_2_2
16	avg_pool_2_2
17	conv_3_1_128
18	max_pool_2_2
19	no_op
20	max_pool_2_2

Figure 20: Network Architecture for network #2

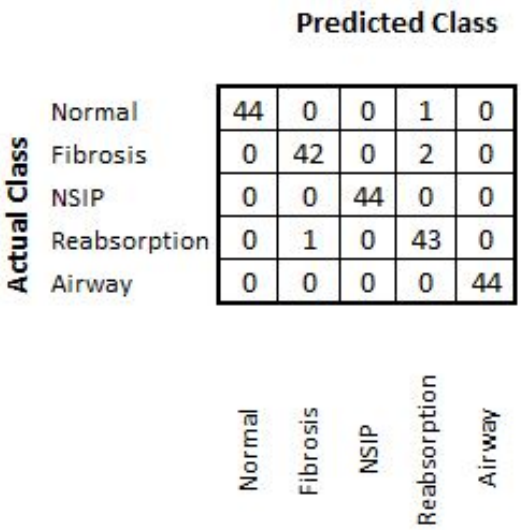

Figure 21: Confusion Matrix for network #2

The confusion matrix in Figure 21 shows what we expected, the classes don't have much trouble in classifying the images and they don't make many mistakes, proving that without the class Other the Q-NAS model can find a network with accuracy close to 100%.

5 Conclusions & future work

In this work we were able to evolve different networks using the Q-NAS model to classify different types of post-covid lung patterns. Using the dataset with 6 classes we observed the impact of different Q-NAS's parameters in the test accuracy, and were able to find a good network with the final accuracy of 90.909%.

We also created two new datasets in order to improve the accuracy and the new networks found for these datasets had a high accuracy for all of the runs. For the dataset with the class Normal combined with Other, our best network had accuracy of 94.091% and for the dataset without the Other class, the best network reached an accuracy of 98.636%.

In future works, we can use these networks to classify CT images from different patients to see how they would work in real life situations. We can also see if the accuracy for the dataset with 6 classes could be improved if the Q-NAS had access to a dataset with more images.

References

[1] I. P. de Sousa, "Inteligência artificial explicável para classificadores de imagens médicas," Ph.D. dissertation, Pontifícia Universidade Católica do Rio de Janeiro, 2021.

[2] H. Iba, *Evolutionary Approach to Machine Learning and Deep Neural Networks: Neuro-Evolution and Gene Regulatory Networks*. Springer Singapore, 2018.

[3] D. Szwarcman, "Quantum-inspired neural architecture search," Ph.D. dissertation, Pontifícia Universidade Católica do Rio de Janeiro, 2020.

[4] "World health organization-coronavirus," last accessed 02/06/2022. [Online]. Available: https://www.who.int/health-topics/coronavirus

[5] C. Sohrabi, Z. Alsafi, N. O'Neill, M. Khan, A. Kerwan, A. Al-Jabir, C. Iosifidis, and R. Agha, "World health organization declares global emergency: A review of the 2019 novel coronavirus (covid-19)," *International Journal of Surgery*, vol. 76, pp. 71–76, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1743919120301977

[6] "World health organization," last accessed 02/06/2022. [Online]. Available: https://covid19.who.int/

[7] C. G. C. Chantong, "Caracterização dos padrões pulmonares da covid-19 pós-aguda na tomografia computadorizada e testagem de um modelo correspondente de inteligência artificial," Master's thesis, Universidade do Estado do Rio de Janeiro, 2022.

[8] M. Mitchell, "Artificial intelligence: A guide for thinking humans," Farrar, Straus and Giroux ,2019.

[9] Y. Bengio, "Deep learning of representations: Looking forward," in *Statistical Language and Speech Processing*, A.-H. Dediu, C. Martín-Vide, R. Mitkov, and B. Truthe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–37.

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

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

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.

[13] F. Chollet, *Deep Learning with Python*. New York: Manning Publications, 2018.

[14] T. Tieleman and G. Hinton, *Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural Networks for Machine Learning, 4, 2012.

[15] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd." 2017, arXiv preprint arXiv:1712.07628.

[16] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, p. 400–407, 1951.

[17] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," 2015, arXiv preprint arXiv:1412.6980.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, arXiv preprint arXiv:1502.03167.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, arXiv preprint arXiv:1512.03385.

[20] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," 2019, arXiv preprint arXiv:1809.00846.

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

[22] J. D. Noce, "Enhanced q-nas for image classification," Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2022.

[23] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Springer, 2015.

[24] M. D. Platel, S. Schliebs, and N. Kasabov, "Quantum-inspired evolutionary algorithm: A multimodel eda," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 6, pp. 1218–1232, 2009.

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

A Appendix

a COVID class in Q-NAS for dataset with 6 classes

```
class COVIDInfo(object):
    def __init__(self, data_path, validation=True):
        self.data_path = data_path
        self.height = 256
        self.width = 256
        self.num_channels = 3

        self.num_classes = 6
        self.pad = 4
        self.train_files = [os.path.join(self.data_path, f) for f in os.listdir(self.data_path)
                            if f.startswith('train')]
        self.valid_files = [os.path.join(self.data_path, f) for f in os.listdir(self.data_path)
                            if f.startswith('valid')]
        self.test_files = [os.path.join(self.data_path, f) for f in os.listdir(self.data_path)
                           if f.startswith('test')]
        if not validation:
            self.train_files = self.train_files + self.valid_files
            self.valid_files = []

        self.num_train_ex = count_records(self.train_files)
        self.num_valid_ex = count_records(self.valid_files)
        self.num_test_ex = count_records(self.test_files)
```

Adapted from [3].