Pontifícia Universidade Católica
do Rio de Janeiro

Guilherme Baldo Carlos

Quantum-inspired Neural Architecture Search applied to Semantic Segmentation

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 2023

Pontifícia Universidade Católica do Rio de Janeiro

Guilherme Baldo Carlos

Quantum-inspired Neural Architecture Search applied to Semantic Segmentation

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:

Rio de Janeiro, April the 11th, 2023

Guilherme Baldo Carlos

Graduated in Electrical Engineering at the Federal University of Espírito Santo (Ufes) in 2019.

Bibliographic data

Baldo Carlos, Guilherme

Quantum-inspired Neural Architecture Search applied to Semantic Segmentation / Guilherme Baldo Carlos; advisor: Marley Maria Bernardes Rebuzzi Vellasco; co-advisor: Karla Tereza Figueiredo Leite. – 2023.

81 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2023.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Engenharia de Materiais – Teses. 3. Busca por Arquitetura Neural. 4. Algoritmos Evolucionários. 5. Segmentação Semântica. 6. Computação com Inspiração Quântica. I. Rebuzzi Vellasco, Marley Maria Bernardes. II. Leite, Karla Tereza Figueiredo. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. IV. Título.

CDD: 621.3

Acknowledgments

First, I would like to express my deepest gratitude to my advisor Marley for her guidance, support, and availability. Since the application process and during the whole master's, Marley has been very supportive and provided valuable feedback and encouragement.

I am also thankful for my co-advisor Karla who was always ready to contribute with great ideas and critical viewpoints.

I would like to mention, as well, the fruitful partnership with other Graduate colleagues that was essential during the whole process.

Thanks should also go to my parents, who are the reason I was able to study and pursue the master's degree. And special thanks to my girlfriend Clara who was always there for me and encouraged me during the whole process.

Last but not least, I would like to thank CAPES for the financial support and to PUC-Rio for the tuition exemption scholarship master's degree.

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

Abstract

Baldo Carlos, Guilherme; Rebuzzi Vellasco, Marley Maria Bernardes (Advisor); Leite, Karla Tereza Figueiredo (Co-Advisor). **Quantum-inspired Neural Architecture Search applied to Semantic Segmentation**. Rio de Janeiro, 2023. 81p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Deep neural networks are responsible for great progress in performance for several perceptual tasks, especially in the fields of computer vision, speech recognition, and natural language processing. These results produced a paradigm shift in pattern recognition techniques, shifting the demand from feature extractor design to neural architecture design. However, designing novel deep neural network architectures is very time-consuming and heavily relies on experts' intuition, knowledge, and a trial and error process. In that context, the idea of automating the architecture design of deep neural networks has gained popularity, establishing the field of neural architecture search (NAS). To tackle the problem of NAS, authors have proposed several approaches regarding the search space definition, algorithms for the search strategy, and techniques to mitigate the resource consumption of those algorithms. Q-NAS (Quantum-inspired Neural Architecture Search) is one proposed approach to address the NAS problem using a quantum-inspired evolutionary algorithm as the search strategy. That method has been successfully applied to image classification, outperforming handcrafted models on the CIFAR-10 and CIFAR-100 datasets and also on a real-world seismic application. Motivated by this success, we propose SegQNAS (Quantum-inspired Neural Architecture Search applied to Semantic Segmentation), which is an adaptation of Q-NAS applied to semantic segmentation. We carried out several experiments to verify the applicability of SegQNAS on two datasets from the Medical Segmentation Decathlon challenge. SegQNAS was able to achieve a 0.9583 dice similarity coefficient on the spleen dataset, outperforming traditional architectures like U-Net and ResU-Net and comparable results with a similar NAS work from the literature but with fewer parameters network. On the prostate dataset, SegQNAS achieved a 0.6887 dice similarity coefficient, also outperforming U-Net, ResU-Net, and outperforming a similar NAS work from the literature.

Keywords

Neural Architecture Search; Evolutionary Algorithms; Semantic Segmentation; Quantum-inspired Computin.

Resumo

Baldo Carlos, Guilherme; Rebuzzi Vellasco, Marley Maria Bernardes; Leite, Karla Tereza Figueiredo. **Busca por arquitetura neural com inspiração quântica aplicada a segmentação semântica**. Rio de Janeiro, 2023. 81p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Redes neurais profundas são responsáveis pelo grande progresso em diversas tarefas perceptuais, especialmente nos campos da visão computacional, reconhecimento de fala e processamento de linguagem natural. Estes resultados produziram uma mudança de paradigma nas técnicas de reconhecimento de padrões, deslocando a demanda do *design* de extratores de características para o design de arquiteturas de redes neurais. No entanto, o *design* de novas arquiteturas de redes neurais profundas é bastante demandanteem termos de tempo e depende fortemente da intuição e conhecimento de especialistas, além de se basear em um processo de tentativa e erro. Neste contexto, a idea de automatizar o *design* de arquiteturas de redes neurais profundas tem ganhado popularidade, estabelecendo o campo da busca por arquiteturas neurais (NAS - Neural Architecture Search). Para resolver o problema de NAS, autores propuseram diversas abordagens envolvendo o espaço de buscas, a estratégia de buscas e técnicas para mitigar o consumo de recursos destes algoritmos. O Q-NAS (*Quantum-inspired Neural Architecture Search*) é uma abordagem proposta para endereçar o problema de NAS utilizando um algoritmo evolucionário com inspiração quântica como estratégia de buscas. Este método foi aplicado de forma bem sucedida em classificação de imagens, superando resultados de arquiteturas de *design* manual nos conjuntos de dados CIFAR-10 e CIFAR-100 além de uma aplicação de mundo real na área da sísmica. Motivados por este sucesso, propõe-se nesta Dissertação o SegQNAS (*Quantum-inspired Neural Architecture Search applied to Semantic Segmentation*), uma adaptação do Q-NAS para a tarefa de segmentação semântica. Diversos experimentos foram realizados com objetivo de verificar a aplicabilidade do SegQNAS em dois conjuntos de dados do desafio *Medical Segmentation Decathlon*. O SegQNAS foi capaz de alcançar um coeficiente de similaridade *dice* de 0.9583 no conjunto de dados de baço, superando os resultados de arquiteturas tradicionais como U-Net e ResU-Net e atingindo resultados comparáveis a outros trabalhos que aplicaram NAS a este conjunto de dados, mas encontrando arquiteturas com muito menos parãmetros. No conjunto de dados de próstata, o SegQNAS alcançou um coeficiente de similaridade *dice* de 0.6887 superando a U-Net, ResU-Net e o trabalho na área de NAS que utilizamos como comparação.

Palavras-chave

Busca por Arquitetura Neural; Algoritmos Evolucionários; Segmentação Semântica; Computação com Inspiração Quântica.

Table of contents

1 Introduction **15**
1.1 Objectives 16
1.2 Contributions 16
1.3 Work outline 17

2 Semantic Segmentation **18**
2.1 Semantic Segmentation 18
2.2 Convolutional Neural Networks 20
2.3 Fully Convolutional Neural Networks 24
2.4 Encoder-Decoder based models 26

3 Neural Architecture Search **31**
3.1 Neural Architecture Search 31
3.2 Search Space 32
3.2.1 Global Search Space 32
3.2.2 Cell Search Space 34
3.3 Search Strategy 35
3.3.1 Reinforcement Learning 36
3.3.2 Evolutionary Algorithms 37
3.3.3 Other methods 39
3.4 Performance Estimation Strategy 40
3.5 Neural Architecture Search applied to Semantic Segmentation 41

4 Q-NAS **45**
4.1 Overview 45
4.2 Search Space and Chromosome Representation 46
4.2.1 Numerical quantum chromosome 46
4.2.2 Categorical quantum chromosome 48
4.3 Quantum Inspired Evolutionary Algorithm Search Strategy and Evaluation Strategy 50

5 SegQNAS **53**
5.1 Search Space 53
5.1.1 Topology level search space 53
5.1.2 Cell level search space 55
5.1.3 Network template and Function set 57
5.2 Decoding 58
5.3 Performance Estimation Strategy 59

6 Experiments and Results **61**
6.1 Datasets 61
6.1.1 Spleen Dataset 61
6.1.2 Prostate Dataset 61
6.2 Setup 62

6.3 Experiments 62
6.3.1 Experiment 1 - Topology Search 63
6.3.2 Experiment 2 - Cell Search 63
6.3.3 Experiment 3 - Bi-level Search 64
6.3.4 Experiment 4 - Full Search 64
6.4 Results and discussions 65
6.4.1 Spleen Dataset 65
6.4.2 Prostate Dataset 68

7 Conclusions 71

Bibliography 73

List of figures

Figure 2.1 © IEEE 2019 Image segmentation: Given an (a) input image we see the ground truth for the given tasks: (b) semantic segmentation (pixel-level labels); (c) instance segmentation (object-level label); (d) panoptic segmentation (both). Source: Kirillov et al. 2019 [6]. 19
Figure 2.2 A typical CNN block comprises a convolutional layer, activation layer, and pooling layer. 21
Figure 2.3 Example of a convolution with $s = 1$ and $p = 0$. This is also called a valid convolution. 22
Figure 2.4 Example of an 2x2 average pooling operation with $s = 2$. 23
Figure 2.5 Typical Classification CNN. The CNN blocks (blocks convolution + activation followed by pooling) extract features from the input image. The fully connected layers use those features as input in order to produce the dense prediction. 23
Figure 2.6 VGG-16. The architecture from the VGG family with 16 layers [29]. 24
Figure 2.7 Example of a transpose convolution with kernel 2×2, input tensor 2×2 producing an output tensor of 3×3. 25
Figure 2.8 Feature maps of a semantic segmentation FCN based on the VGG-16 architecture. 25
Figure 2.9 Feature maps of a semantic segmentation FCN based on the VGG-16 architecture. 26
Figure 2.10 Segmentation mask produced by FCN-32s, FCN-16s and FCN-8 compared to the ground truth [33]. 27
Figure 2.11 Default autoencoder representation. 27
Figure 2.12 The max unpooling is an example of an unpooling layer. The max unpooling uses the indices of the max elements in the pooling operations in the encoder stage to define the position of the unpooled elements in the output. Other examples of unpooling layers include the nearest neighbor unpooling and the bed of nails. 28
Figure 2.13 SegNet layer representation. 29
Figure 2.14 U-Net architecture. 30

Figure 3.1 NAS process. The search strategy samples candidate architectures from the search space. The candidates are evaluated accordingly to a performance estimation strategy. 31
Figure 3.2 Chain-structured search space. 33
Figure 3.3 Multi-branch search space. 34
Figure 3.4 Templates proposed for the NASNet search space (a) template proposed for CIFAR10 (b) template proposed for ImageNet. The green cells are fixed cells in the template. 36
Figure 3.5 A general framework for RL methods. 37
Figure 3.6 A general framework for EA methods. 38

Figure 4.1 Q-NAS overview. The user must predefine the network template, function set, hyperparameters, and ranges. Q-NAS produces as output the best architecture and its hyperparameters. 46
Figure 4.2 Q-NAS simplified flowchart showing in the dark boxes the steps are executed differently for the two parts of the chromosome. 47
Figure 4.3 The observation process generates a classical individual from the quantum individual. The classical individual is then decoded into the network to be evaluated. 49

Figure 5.1 One possible network in the topology search space. The circles represent feature maps, while the arrows represent the cells. The dark grey circles represent the network's input (left) and the output (right). The yellow circles are feature maps that are outputs of the network's cells. The first and the last arrows represent, respectively, a stem convolution to adequate the feature map shape and the final convolution that produces the segmentation maps. The solid arrows are cells of the network, and the dashed arrows represent other possibilities in the search space; therefore, feature maps that are not connected by any arrow are infeasible. Skip connections are omitted for illustration purposes. The cell level is on the right ranging from 0 to 4 ($D = 4$), and on the top, the cell enumeration ranges from 0 to 9 ($L = 10$). 54
Figure 5.2 U-Net topology represented in SegQNAS search space. The dashed arrows connecting the encoder and decoder are the skip-connections. 55
Figure 5.3 Cell structure. The dashed blocks exist only when applicable. The *concat* block exists only when there is a skip connection to that cell. The upscaling/downscaling blocks exist depending on the cell type. 56
Figure 5.4 Blocks of SegQNAS. Conv(kxk) refers to a $k \times k$ convolution where k is also searchable. BN stands for Batch Normalization and ReLU indicates all activations as ReLU. All blocks produce as output a feature map with the same shape as the input. 57
Figure 5.5 Network from SegQNAS search space. 57
Figure 5.6 Operations that lead to infeasible networks in red. Those operations disrespect the design constraints listed in Section 5.1. 58

Figure 6.1 Classical individual and decoded network for experiment 1 on the spleen dataset. 66
Figure 6.2 Classical individual and decoded network for experiment 2 on the spleen dataset. 67
Figure 6.3 Classical individual and decoded network for experiment 3 on the spleen dataset. 67
Figure 6.4 Classical individual and decoded network for experiment 4 on the spleen dataset. 67

Figure 6.5 Classical individual and decoded network for experiment 1 on the prostate dataset. 69
Figure 6.6 Classical individual and decoded network for experiment 2 on the prostate dataset. 69
Figure 6.7 Classical individual and decoded network for experiment 3 on the prostate dataset. 70
Figure 6.8 Classical individual and decoded network for experiment 4 on the prostate dataset. 70

List of tables

Table 6.1 Summary of the experiments 64
Table 6.2 Spleen Dataset Experiments Results 65
Table 6.3 Spleen Dataset Experiments Results comparison with MixedBlock [85] 66
Table 6.4 Prostate Dataset Experiments Results 68
Table 6.5 Prostate Dataset Experiments Results comparison with MixedBlock [85] 68

List of Abreviations

ANN	Artificial Neural Network
ASPP	Atrous Spatial Pyramid Pooling
CNN	Convolutional Neural Network
CRF	Conditional Random Field
DAG	Direct Acyclical Graph
DNN	Deep Neural Network
DPC	Dense Prediction Cell
EA	Evolutionary Algorithm
FCN	Fully Convolutional Network
MLP	Multilayer Perceptron
NAS	Neural Architecture Search
QIEA	Quantum-Inspired Evolutionary Algorithm
Q-NAS	Quantum-Inspired Neural Architecture Search
RL	Reinforcement Learning
SegQNAS	Quantum-inspired Neural Architecture Search applied to Semantic Segmentation

1
Introduction

Deep learning techniques have achieved excellent results in various perceptual tasks in computer vision, speech recognition, and natural language processing. This success is owed to the capacity these methods have to automatically learn to extract features from unstructured data [1].

Traditional pattern recognition methods relied on handcrafted feature extractors, which were time-consuming and lacked efficacy. Deep learning techniques, on the other hand, are able to automatically learn to extract features that are important for the task and that may be non-obvious even to an expert in the field. Besides that, deep learning techniques are able to extract those multiple features on multi scales, in a hierarchical form, and define how important each of those features is for the task.

These methods are based on Deep Neural Network (DNN) models, which are Artificial Neural Networks (ANN) composed of multiple layers. Despite the capacity to automatically learn to extract features, several hyperparameters must be manually defined to achieve good results with DNNs. Each layer of the network can perform a different operation, i.e., convolution, fully connected layers, pooling, transposed convolution, etc. Furthermore, every layer could have hyperparameters to define. For example, for a convolutional layer, one should define the number of kernels, kernel size, stride, and padding. Moreover, the number of layers and how the layers are interconnected have also to be defined.

Historically the architecture definition of DNNs is manually done, relying on experts' knowledge and a trial and error process. However, manually designing novel DNN architectures, aside from being time-consuming, is prone to human bias and may leave non-obvious architectures unexplored.

A logical step in the direction of automatizing deep learning methods is to automatize the architecture definition of the DNNs used. This research field is known as Neural Architecture Search (NAS) [2].

Several methods have been proposed in the literature to tackle the NAS problem. Q-NAS (Quantum-inspired Neural Architecture Search [3] is one of the proposed methods, which introduces a Quantum-inspired Evolutionary Algorithm (QIEA) to perform the search for a deep neural architecture on the image classification task. Q-NAS was able to outperform hand-designed models from the literature on the CIFAR-10 and CIFAR-100 datasets, and also on a real-world seismic application.

The majority of papers on NAS apply their work to image classification. Motivated by this and by the success of Q-NAS, this work extends Q-NAS to Semantic Segmentation, which is another fundamental task in the computer vision field.

1.1 Objectives

In this work, we introduce SegQNAS (Quantum-inspired Neural Architecture Search applied to Semantic Segmentation), which is an extension of Q-NAS to perform NAS on semantic segmentation DNNs.

Our primary goal is to propose a NAS algorithm based on Q-NAS to search for semantic segmentation deep neural networks.

The secondary goal of this work is to evaluate the performance of SegQNAS on two datasets from the Medical Segmentation Decathlon challenge [4] and compare its performance to well-established baselines from the literature and a similar NAS work from the literature.

1.2 Contributions

In this section, we list the main contributions of this work:

– **Extending Q-NAS to semantic segmentation**. Semantic segmentation plays a major role in a variety of applications. Therefore, in this work, we extended Q-NAS to semantic segmentation.

– **The usage of a bi-level search space**. An adequate search space had to be used to extend Q-NAS to semantic segmentation. The search space is a global search space as in the original Q-NAS. However, a bi-level search space accounts for both topology and cell levels search. That approach allows for performing the search either separately on the topology and cell levels, sequentially, or simultaneously.

– **Decoding rules to ensure feasibility**. In this work, we propose some decoding rules to ensure the decoded network is a feasible segmentation network according to some design constraints.

1.3 Work outline

This work comprises 6 additional chapters, which we describe below.

In Chapter 2, we present the semantic segmentation task and describe how current deep neural network architectures are designed to tackle this computer vision problem.

Next, Chapter 3 provides the theoretical background to understand NAS, and its components, namely search space, search strategy, performance estimation strategy, and also its challenges.

Chapter 4 presents the Q-NAS model, its individual representation, specific operations, algorithmic steps, and implementation details.

SegQNAS is introduced in Chapter 5. In this chapter, we discuss the changes made to apply Q-NAS to semantic segmentation, including the search space used and decoding strategies to ensure the feasibility of the networks decoded. Furthermore, we discuss the techniques used to mitigate the time consumption of the search.

Chapter 6 describes and discusses the experiments proposed to validate SegQNAS. The results are presented and compared to baselines and other models in the literature.

Finally, Chapter 7 concludes our work and discusses the next steps of our research.

2
Semantic Segmentation

In this chapter, we present what is semantic segmentation and how deep learning methods based on convolutional neural networks are used to solve this task. Firstly, a brief review is provided of how convolutional neural networks are able to perform automatic feature extraction learning. Then, fully convolutional networks are presented. Architectures like FCN, DeconvNet, SegNet, and U-Net and their particularities are discussed.

2.1
Semantic Segmentation

Image segmentation is a fundamental computer vision task. Its goal is to partition images or video frames into semantic labeled regions [5]. This task can be broken down into three classes of problems.

- *Semantic segmentation*: This can be formulated as the task of pixel-level classification, assigning a semantic label to every pixel in the image.
- *Instance segmentation*: In contrast, instance segmentation involves partitioning individual instances of objects in the image.
- *Panoptic segmentation*: Panoptic segmentation deals with performing both semantic and instance segmentation.

As can be seen in Figure 2.1, image *a* shows an example of an input image, and in images *b*, *c*, and *d* the ground truth for semantic segmentation, instance segmentation, and panoptic segmentation, respectively. It is possible to notice that in image *b*, every pixel of the image is classified into a semantic label, not differentiating instances of objects. Therefore, every pixel of a *car* in the image *a* is mapped to the *car* label. The same is true for the pixels of the *sky*, *person*, *street*, *sidewalk*, and so on. On the other hand, in image *c*, each instance of *person* or *car* is mapped to a different semantic label. This characterizes the instance segmentation problem. Finally, in image *d*, we see what the ground truth of panoptic segmentation looks like. In that case, both semantic segmentation and instance segmentation are performed, so besides classifying every pixel of the image with a semantic label, one should also

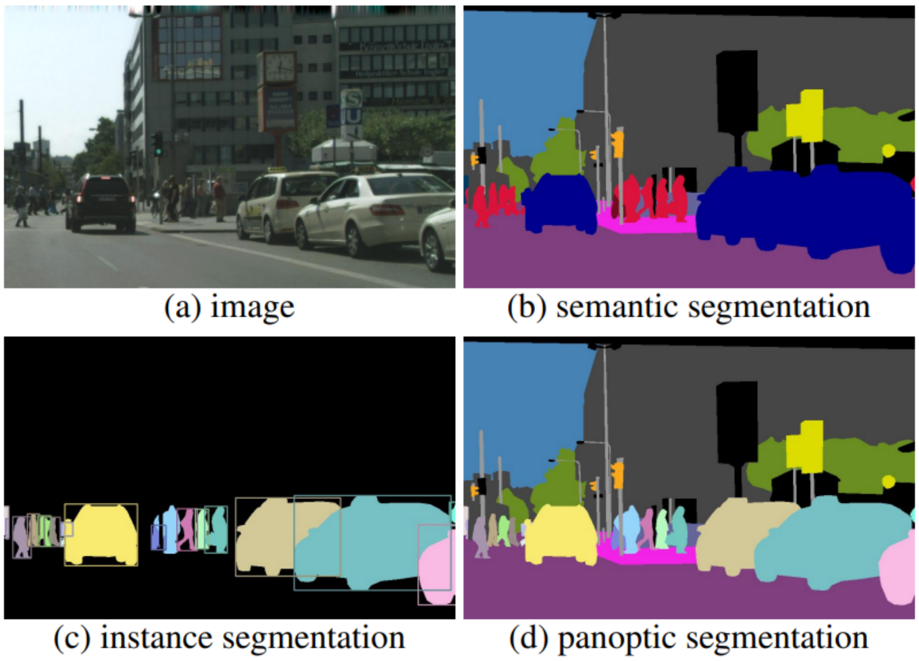
(a) image
(b) semantic segmentation
(c) instance segmentation
(d) panoptic segmentation

Figure 2.1: © IEEE 2019 Image segmentation: Given an (a) input image we see the ground truth for the given tasks: (b) semantic segmentation (pixel-level labels); (c) instance segmentation (object-level label); (d) panoptic segmentation (both). Source: Kirillov et al. 2019 [6].

differentiate between different instances of objects. In this work, we are going to focus on the problem of semantic segmentation.

Semantic segmentation plays a major role in a broad domain of real-world applications, including scene parsing [7], autonomous driving [8], remote sensing [9], video surveillance [10], medical image analysis [11], among others. When it comes to autonomous driving applications, the labels could be road, pedestrian, vehicle, etc. On the other hand, when it comes to medical image analysis, the semantic labels could be organs, vessels, and tissues, to name a few possibilities.

As described, semantic segmentation is the task of classifying an input image on a pixel level. That is, every pixel of the input image should be mapped to a semantic label belonging to a set of predefined labels. Thus, this is generally a more demanding undertaking task than image classification, which deals with attributing a unique label to the whole image [5]. Due to the nature of the task, semantic segmentation methods should be able to extract features from the image that have not only semantic information but also localization information.

In order to do that, several semantic segmentation methods have been developed in the literature, such as thresholding [12], region-growing [13], k-means clustering [14], watershed methods [15], active contours [16], graph cuts [17], conditional random fields (CRFs) [18], Markov random fields [19] and sparsity-based methods [20, 21]. One challenge these feature-based methods face is that they rely heavily on the quality of the features extracted by domain experts, although humans are bound to miss latent or abstract features that could be important to the problem [22].

In that sense, Deep Learning methods were able to tackle the issue of automated feature learning, achieving remarkable performance in several computer vision tasks, such as classification, object detection, and semantic segmentation [22]. However, it is important to notice that the success of those methods is highly supported by the large data and computing resource availability we are witnessing.

The next sections will present the basic concepts of Convolutional Neural Networks, how they extract features from an input image, and how some architectures designed for Semantic Segmentation work.

2.2 Convolutional Neural Networks

In 1980 Fukushima [23] introduced an artificial neural network based on a hierarchical model of the visual system's structure. His network, called Neocognitron, was composed of several layers and each layer was composed of neurons sharing parameters. The neurons received as input patches of the previous layer, producing translational invariance. That aspect is very important for recognition tasks, where the goal is to recognize a pattern despite its location in the image. The Neocognitron was trained on an unsupervised scheme. Later in 1989, Yan Lecun and other collaborators proposed the Convolutional Neural Network (CNN) [24] building upon the Neocognitron's ideas. However, the CNN was trained with back-propagation and therefore trained in a supervised manner, achieving state-of-the-art results for pattern recognition tasks [25].

CNNs are deep neural networks, meaning that they are composed of multiple layers. Typically they are composed of stacked blocks of convolutional layers, activation layers, and pooling layers (Figure 2.2).

A convolutional layer receives as input a tensor with shape $(H \times W \times C)$ where H is the height, W is the width, and C is the number of channels of the input tensor. Then, it convolves k kernels (or filters) of size $(h \times w \times C)$ along the x and y axis of the input tensor, producing an output tensor with shape $(\frac{H-h+2\times p}{s} + 1 \times \frac{W-w+2\times p}{s} + 1 \times k)$ where p is the padding and s is the stride. Padding is the number of pixels added around the input tensor and stride is

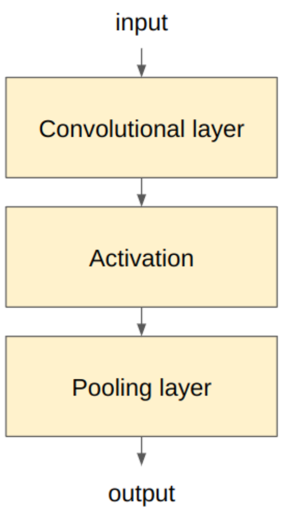

Figure 2.2: A typical CNN block comprises a convolutional layer, activation layer, and pooling layer.

the number of units the kernel slides on the input tensor when performing the convolution. Those parameters are used to control the output tensor shape. Figure 2.3 illustrates the convolution operation applied to a $4 \times 4 \times 1$ input tensor with a $2 \times 2 \times 1$ kernel with $s = 1$ and $p = 0$ producing an output tensor of $3 \times 3 \times 1$.

The kernel is a tensor of weights and during training, these weights are adjusted using the back-propagation algorithm to extract task-specific features from the input tensor. Therefore, the output tensor of a layer is called a feature map.

The use of convolutional layers provides three main advantages to neural networks when compared to fully connected layers, traditionally used in multi-layer perceptrons (MLP) [26]:

– *Sparse interactions*: In fully connected layers, all of the neurons are connected to all of the previous layer neurons. Therefore, each element in the input influences each element in the output of the layer. In contrast, in a convolutional layer, each input element influences only a patch of the output element. As can be seen in Figure 2.3, The element b in the input tensor is only affecting the first and second elements of the output tensor. The sparse connectivity reduces memory usage and the number of operations required to produce the output. This is translated into a significant efficiency improvement.

– *Parameter sharing*: As the convolution operation uses the same kernel to convolve through the whole input tensor, the parameters are shared.

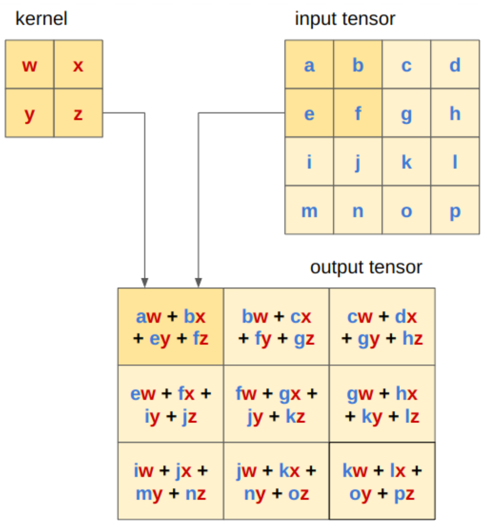

Figure 2.3: Example of a convolution with $s = 1$ and $p = 0$. This is also called a valid convolution.

This also produces an efficiency improvement.

– *Translational invariance*: This means that the kernels can identify the same features regardless of their position in the input tensor. However, the convolution operation is not invariant to other transformations like scale and rotations. This invariance can be achieved during training by applying data augmentation techniques.

The second type of layer is the activation layer. This type of layer performs an element-wise non-linear operation on an input tensor, enabling the CNN to learn non-linear features. There are several possibilities for activation functions but the most common is ReLU [27], which is defined as $f(x) = max(0, x)$.

Finally, pooling layers reduce the spatial resolution of an input tensor. Given a patch of the input of size $k \times k$, the pooling layer replaces the values by an aggregation statistic measure like average or max. That patch also slides through the input tensor with stride s. Figure 2.4 illustrates a 2×2 average pooling operation.

The dimensional reduction in the feature maps is important for two main reasons. First, reducing the feature maps resolution can reduce the resource requirement, which improves efficiency; second, it allows for small kernels to have a larger receptive field. That means that a larger region of the input image is taken into account in a kernel patch during convolution. When we stack CNN blocks (Figure 2.2) in a chain-like structure, earlier layers extract more localized features, while deeper layers are able to extract more semantic information.

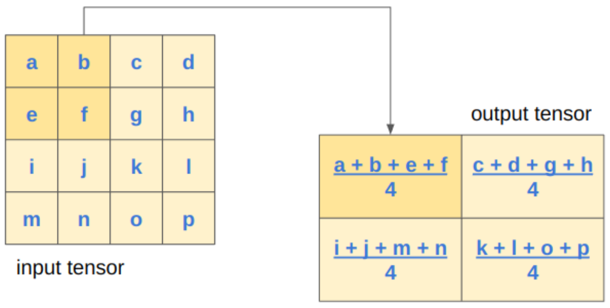

Figure 2.4: Example of an 2x2 average pooling operation with $s = 2$.

A classification CNN is formed by stacking several CNN blocks (Figure 2.5) followed by fully connected layers. The fully connected layers are responsible for receiving the features extracted by the CNN blocks and outputting an array of size $N \times 1$, containing the probability distribution over N classes.

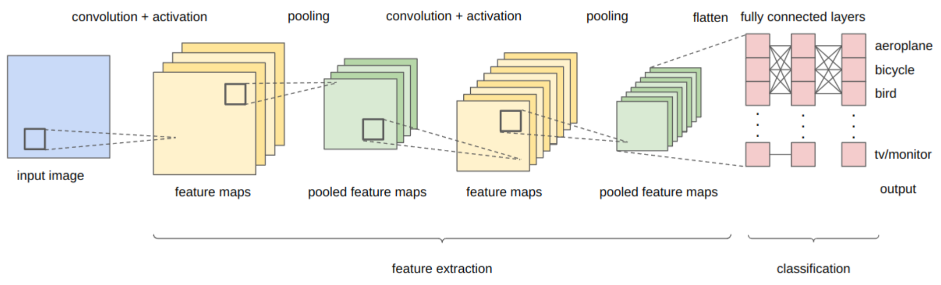

Figure 2.5: Typical Classification CNN. The CNN blocks (blocks convolution + activation followed by pooling) extract features from the input image. The fully connected layers use those features as input in order to produce the dense prediction.

Since the introduction of CNNs, several architectures have been proposed, such as LeNet [25], AlexNet [28], VGGNet [29], InceptionNet [30], ResNet [31], DenseNet [32] and so on. In addition, all of them included improvements on hyperparameters and architecture over the previous ones.

In Figure 2.6 we see the architecture of VGG-16, one architecture from the VGG family [29]. In that Figure, the activation function (ReLU) is implicit in the convolutional layers. It is possible to notice that the main idea is that we have blocks of convolutional layers followed by pooling layers. The number of kernels increases with the depth of the network, and the resolution of the feature maps decreases as pooling layers are applied. So, despite all convolutions being 3×3, the receptive field increases since the feature maps are downsampled by the pooling layers. That layout allows for features to be hierarchically extracted. Earlier layers extract general localised features like contours, and deeper layers use those features to extract more complex semantic features. The semantically rich features are useful for the fully connected layers to perform classification. However, localisation information is lost during both the feature extraction process and the flattening of the feature maps to be fed to the fully connected layers.

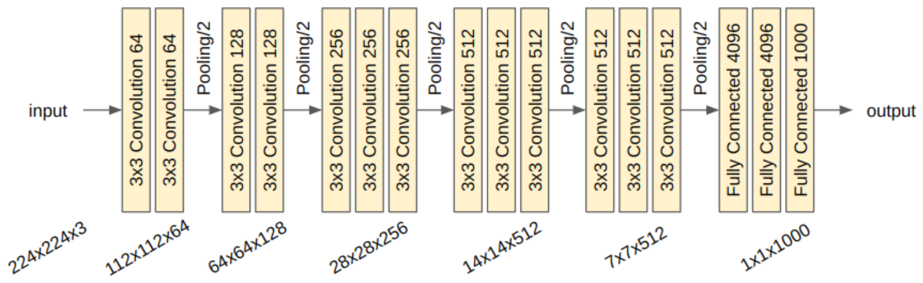

Figure 2.6: VGG-16. The architecture from the VGG family with 16 layers [29].

The next section covers how to adapt the CNN to perform semantic segmentation.

2.3 Fully Convolutional Neural Networks

The typical classification CNN, shown in Figure 2.5, predicts a label for the whole input image. This is suitable for classification but not for semantic segmentation, as we would like to predict a label for every pixel in the input image. To solve that issue, Fully Convolutional Networks (FCNs) were proposed [33]. The authors used well-known classification CNNs architectures like VGG-16 and performed the following modifications:

1. The flattening operation was removed and the fully connected layers were replaced by 1×1 convolutional layers. That allowed the classification CNN to receive input images of arbitrary sizes.

2. In order to output semantic segmentation maps, a transpose convolutional layer is appended to produce a $H \times W \times N$ output (where H and W are the height and width of the input image respectively and N is the number of segmentation classes). The transpose convolution (or deconvolution) is applied in order to increase the spatial resolution of the output feature maps. This operation is shown in Figure 2.7.

In Figure 2.8, the feature maps of the FCN VGG-16 modified to perform semantic segmentation are shown. As can be seen, now the output feature maps

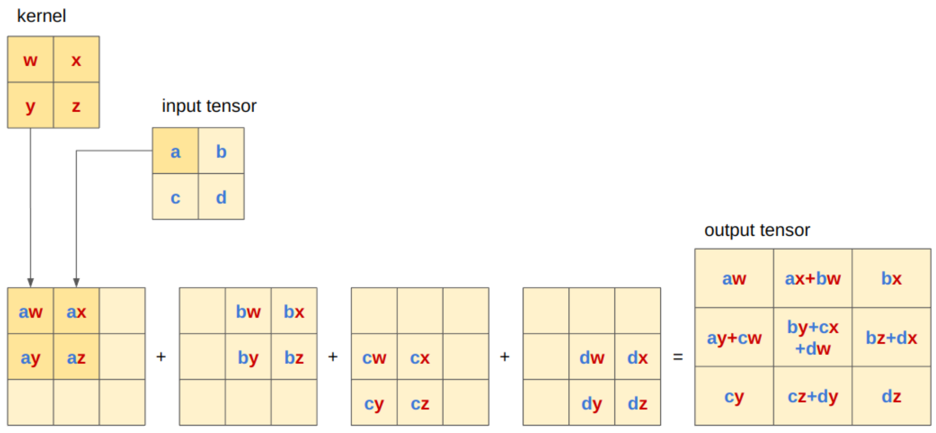

Figure 2.7: Example of a transpose convolution with kernel 2×2, input tensor 2×2 producing an output tensor of 3×3.

(segmentations maps) are the same resolution as the input image but with the number of channels equal to the number of classes, N, in the dataset. However, notice that to generate the segmentation maps, the transposed convolution had to upsample the previous feature map by 32 times, that is, the prediction is made on semantically rich features but with low localisation information, leading to a coarse segmentation.

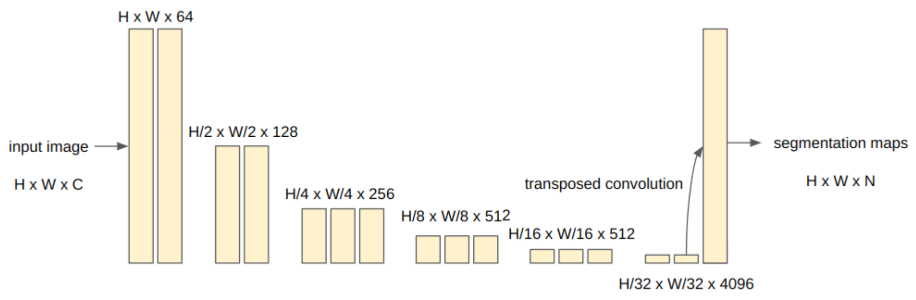

Figure 2.8: Feature maps of a semantic segmentation FCN based on the VGG-16 architecture.

To tackle this issue, the authors proposed combining the feature maps from earlier layers to add localisation information to the prediction and achieve a finer segmentation. They proposed three architectures. First, FCN-32s performed the prediction solely on the last feature map (32-stride). Second, FCN-16s upsampled 2 times the 32-stride feature map and added it to the 16-stride feature map to perform the prediction. Lastly, FCN-8s upsampled 4 times the 32-stride feature map, 2 times the 16-stride feature map and added them to the 8-stride feature map in order to perform the prediction. The upsampling and the predictions are performed by transposed convolutions.

Figure 2.9 shows the three architectures.

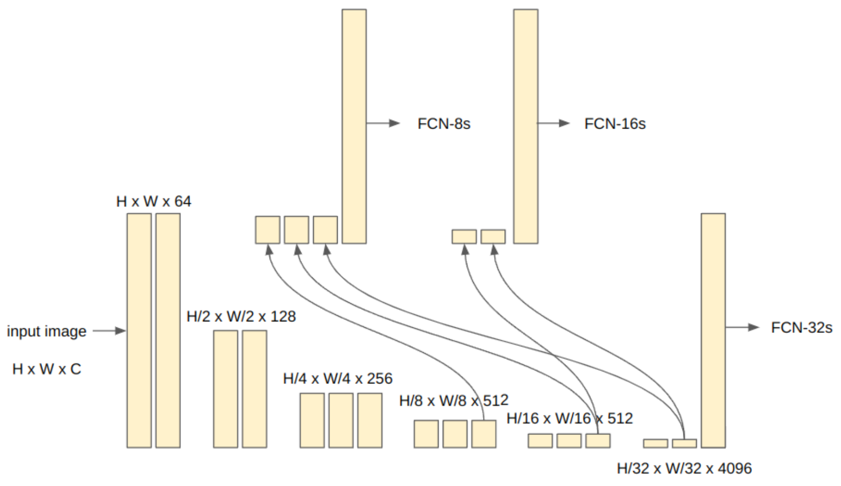

Figure 2.9: Feature maps of a semantic segmentation FCN based on the VGG-16 architecture.

Adding earlier layers to the semantically rich feature maps provided localisation information to the feature map used to produce the segmentation maps. The semantically rich features that are key for classification help to answer the question "what" and the localisation-rich features coming from the earlier layers of the architecture help to answer the question "where". Both types of features are fundamental to the task of semantic segmentation.

The comparison between the segmentation mask produced by FCN-32s, FCN-16s, and FCN-8s is shown in Figure 2.10. It is noticeable that FCN-16s was able to produce a finer segmentation than FCN-32s and that FCN-8s produced an even finer one, supporting the idea that earlier layer feature maps could provide essential features for computer vision tasks that need localisation information.

As seen in Section 2.2, the convolutional blocks of classification CNNs perform very well in extracting semantically important features of the input image. In Section 2.3 we stated that despite the importance of the semantic features, localisation-rich features are also necessary for the task of semantic segmentation as we are interested in answering not only the "what" question but also the "where" question. In the next section, we cover a family of FCNs that exploit the encoder-decoder approach to perform semantic segmentation.

2.4
Encoder-Decoder based models

Traditionally, autoencoders have been used to extract features from input images while retaining most of the original information. That is done with an

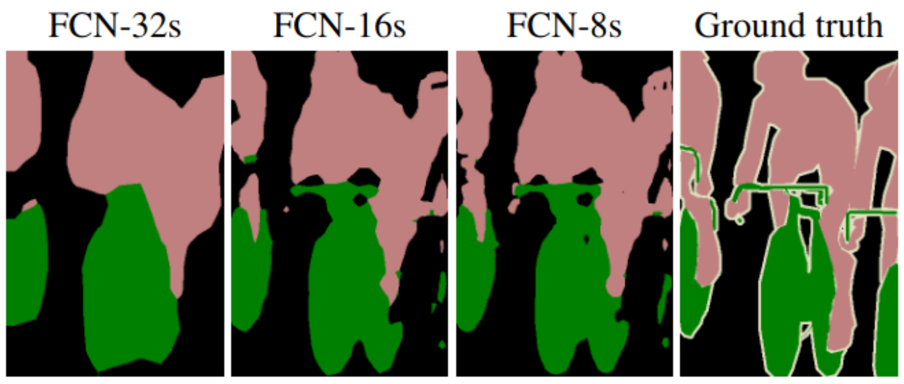

Figure 2.10: Segmentation mask produced by FCN-32s, FCN-16s and FCN-8 compared to the ground truth [33].

encoder-decoder architecture (Figure 2.11) where the encoder is responsible for the feature extraction while the decoder attempts to reconstruct the input from the extracted features. The idea is that the extracted features, also called latent space, in the bottleneck of the autoencoder, have the information necessary for the decoder to reconstruct the input image. For those cases, the loss is calculated by a difference measure between the input and the output of the autoencoder.

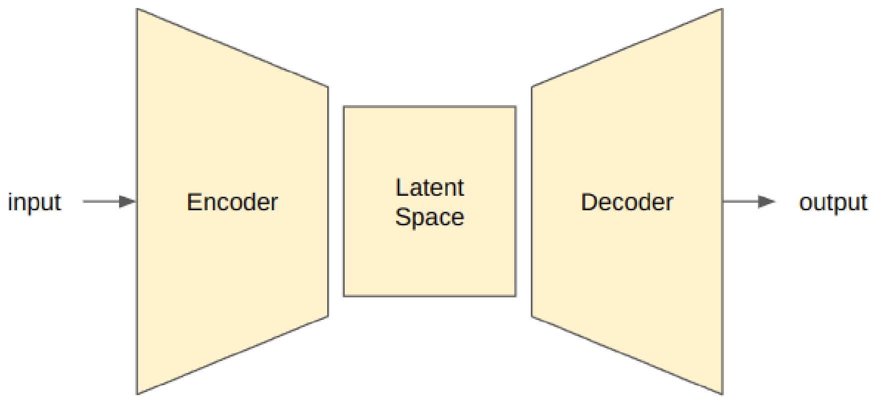

Figure 2.11: Default autoencoder representation.

When applying this idea to perform semantic segmentation, the loss is computed as the difference between the output and the pixel-level annotated ground truth. The generative nature of the decoder is beneficial to generate sharper borders semantic segmentation masks as output. The encoder is like a feature extractor of a typical CNN (Figure 2.5): a series of convolution, activation, and pooling layers, while the decoder uses transposed convolutional layers (Figure 2.7) or unpooling layers (Figure 2.12) in order to generate the segmentation maps from the extracted features.

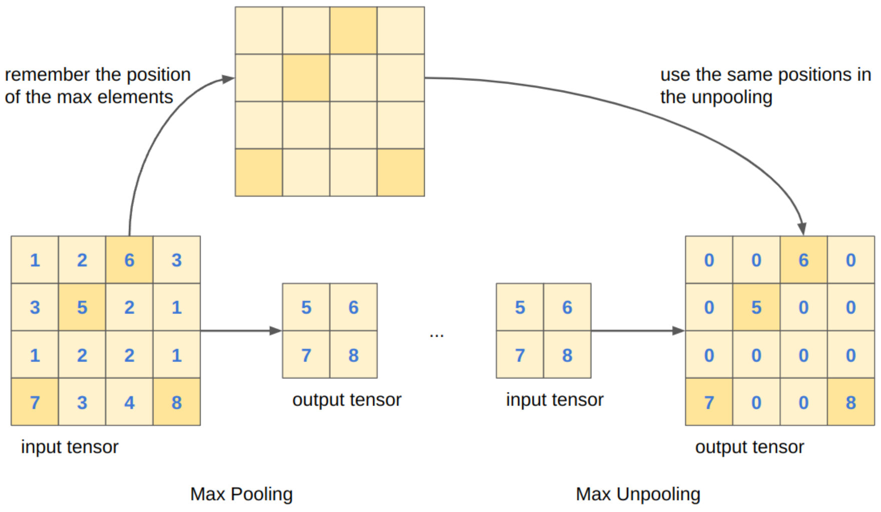

Figure 2.12: The max unpooling is an example of an unpooling layer. The max unpooling uses the indices of the max elements in the pooling operations in the encoder stage to define the position of the unpooled elements in the output. Other examples of unpooling layers include the nearest neighbor unpooling and the bed of nails.

DeconvNet [34] and SegNet [35] are encoder-decoder architectures proposed to perform semantic segmentation using the max unpooling operation (2.12) in the decoder. DeconvNet uses VGG-16 (with fully connected layers replaced by their 1×1 convolutional layer equivalent) (Figure 2.6) as the encoder. As the decoder, the mirrored VGG-16 with max-pooling layers is replaced by max unpooling layers, and convolution operations are replaced by transposed convolution operations, producing at the output the segmentation maps with the same size as the inputs. SegNet (Figure 2.13) proposes a very similar architecture that uses the first 13 convolutional layers of VGG-16 and also uses a mirrored encoder as a decoder with max-pooling layers replaced by max unpooling layers.

For both DeconvNet and SegNet architectures, the decoder uses the output feature map and pooling indices of the encoder in order to produce the segmentation output. U-Net [36] (Figure 2.14), on the other hand, does not use the pooling indices from the encoder but uses the entire feature maps. This is done by adding skip connections from the encoder to the decoder. A skip connection is added after every encoder convolutional block (before the pooling operation) to the decoder. The feature map from the encoder is concatenated with the upsampled feature map from the decoder (transposed convolution output) and used as input for a new convolutional block. Adding

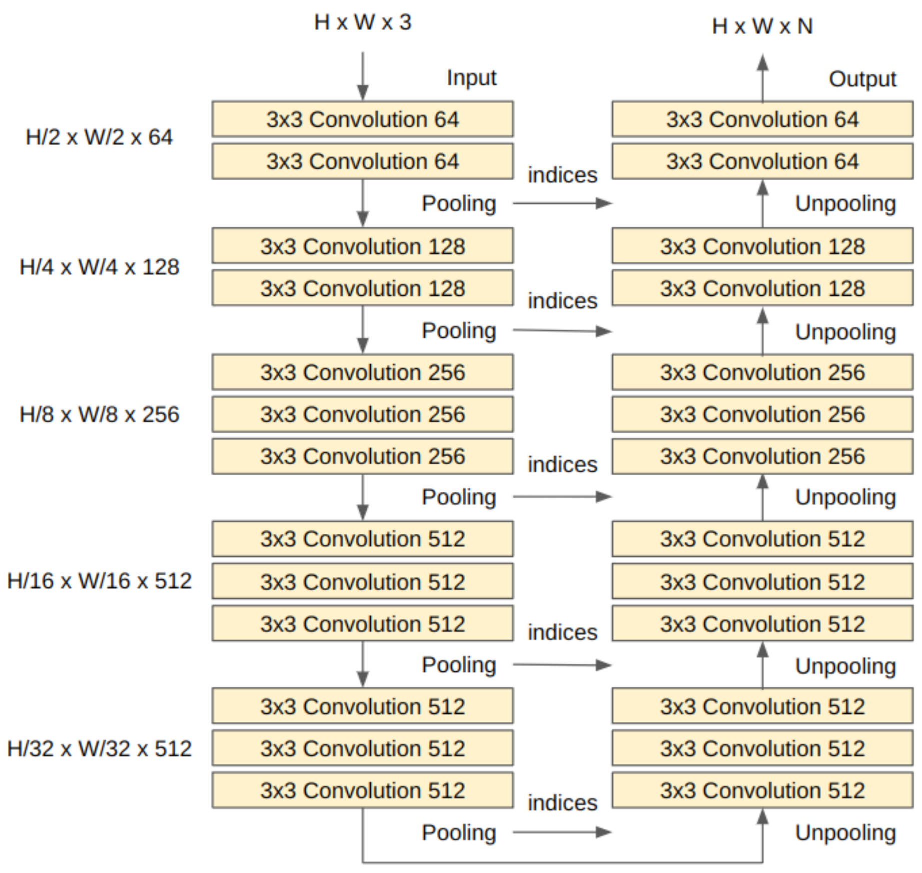

Figure 2.13: SegNet layer representation.

those skip connections to use feature maps from different levels in the decoding process has proved to be very effective to produce sharper segmentation models [22].

Other contributions of the U-Net paper [36] include the training strategy that heavily relies on data augmentation and the Soft Dice loss function based on the Dice coefficient.

U-Net was originally designed to perform semantic segmentation on biological microscopy images. However, it was successful to perform semantic segmentation on a variety of applications. That motivated other works to develop variations of the U-Net architecture like UNet++ [37], V-Net [38] and the One hundred layer Tiramisu [39].

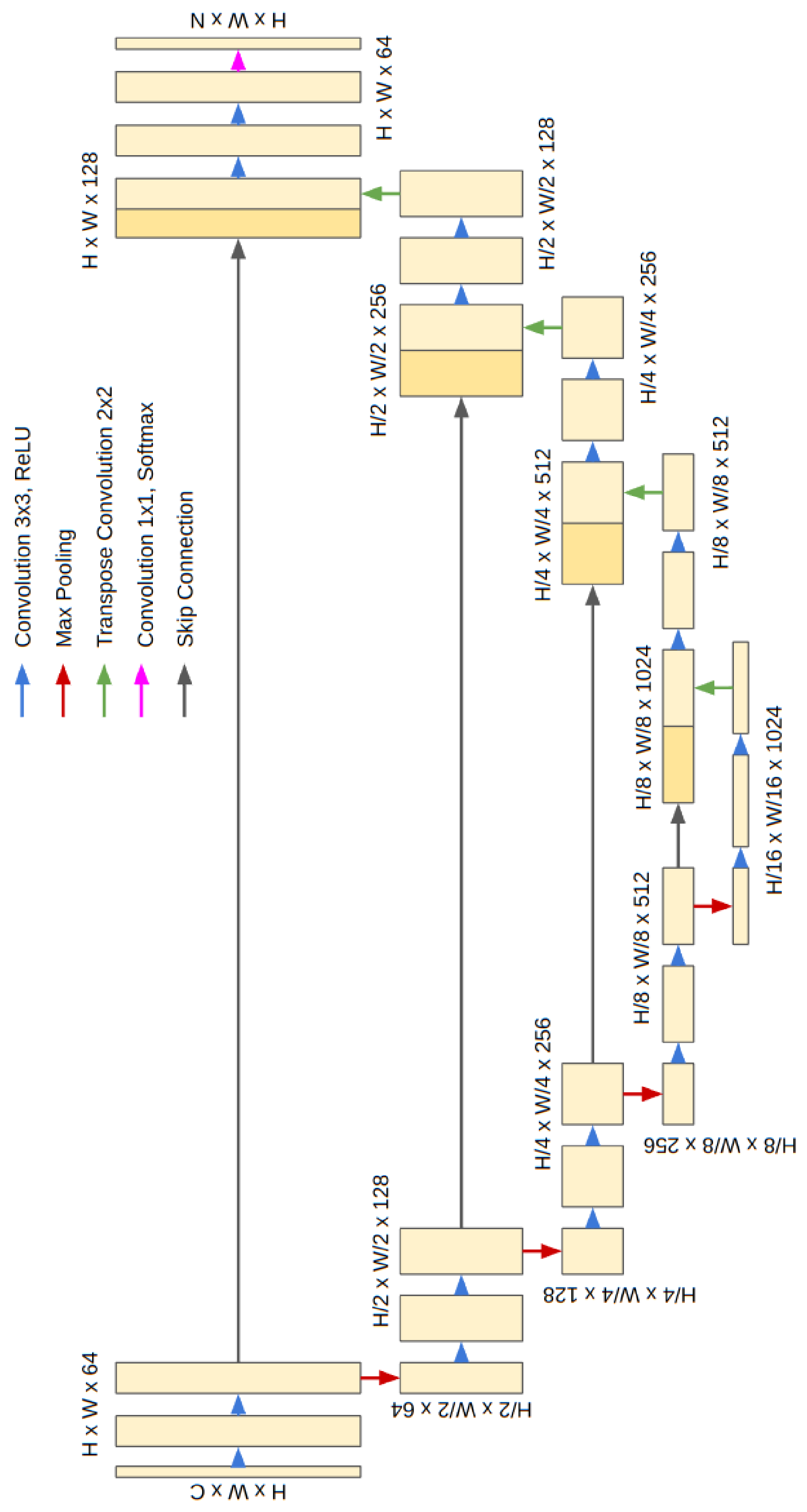

Figure 2.14: U-Net architecture.

3
Neural Architecture Search

In this chapter, we present the field of Neural Architecture Search (NAS). First, we describe its main components: Search space, Search Strategy, and Performance Estimation Strategy. Then we review some of the Neural Architecture Search methods applied to Semantic Segmentation.

3.1
Neural Architecture Search

NAS can be seen as a search process defined by three components: the search space, the search strategy, and the performance evaluation strategy (Figure 3.1) [1]. The search strategy samples candidate architectures - a - from the search space A, and these candidate architectures are evaluated accordingly to a performance estimation strategy. The evaluation is used both for determining which candidates are better than other and to assist in the search strategy sampling of the search space as the problem deal with the exploration-exploitation trade-off.

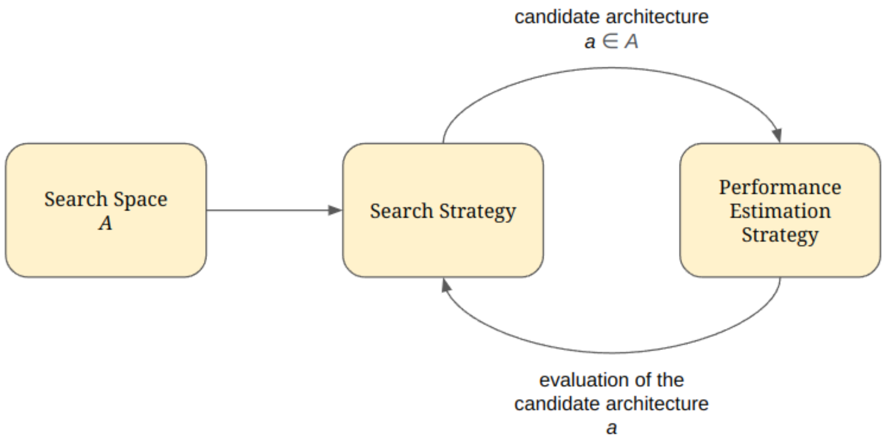
candidate architecture
$a \in A$
Search Space
A
Search Strategy
Performance Estimation Strategy
evaluation of the candidate architecture
a

Figure 3.1: NAS process. The search strategy samples candidate architectures from the search space. The candidates are evaluated accordingly to a performance estimation strategy.

3.2 Search Space

Following the notation used in [40], a DNN can be represented as a direct acyclical graph (DAG) composed by n nodes $\in Z$, where each node $z^{(k)}$ of the graph represents a tensor and its associated operation $o^{(k)} \in O$ that is applied to its set of parent nodes $I^{(k)}$. Input nodes have neither parent nodes nor operations. The computation performed by a node k is then represented by

$$z^{(k)} = o^{(k)}(I^{(k)}) \tag{3-1}$$

The set of operations O can contain unary or multivariate operations. Nodes that perform unary operations can have a single parent. Examples of these operations include convolutions, pooling, activation, and etc. In comparison, nodes that perform multivariate operations can have multiple parent nodes. Examples of multivariate operations are concatenation, addition, and so on. Moreover, the operations O could also contain blocks of layers instead of unique layers.

The set of all possible architectures represented by this DAG comprises the search space. Therefore, it combines all possible ways of connecting the nodes and all possible operations in the O set. An architecture from the search space is denoted as a.

However, the search space can be limited by imposed constraints that insert prior knowledge into the search space and reduce it. These constraints simplify the NAS process, but the loss in the degrees of freedom may prevent the search from exploring promising architectures.

The choice of the search space defines hugely impacts the difficulty of the search problem, since these are non-continuous and with exponentially large search spaces [1].

Focusing on computer vision tasks, and therefore, thinking about search spaces dedicated to representing CNNs, the search spaces can be divided into two main groups: global search space and cell search space. Those search space approaches will be further explained in the next sections.

3.2.1 Global Search Space

Global search spaces allow for a higher degree of freedom in the network definition. Network templates are proposed to guarantee that the search space is not exponentially large and that the DNNs make sense for the desired task. These templates insert constraints into the search space. For example, the number of nodes n in the DAG, how the node $z^{(k)}$ might be connected to the other nodes, the operations o that are allowed in each node $z^{(k)}$, and some fixed layers. For example, for a classification, CNN one might impose the last layers to be fully connected.

A simple chain-structured search space is proposed by [41] (Figure 3.2). In this search space, the number of nodes is fixed to n and the node $z^{(k)}$ has only one parent, the node $z^{(k-1)}$. That way, the node computation described in equation 3-1 is simplified to:

$$z^{(k)} = o^{(k)}(z^{(k-1)}) \tag{3-2}$$

The operations o are all unary and belong to the set $O = \{$convolutions, pooling operations, and fully connected layers$\}$. Additional constraints are taken into account. Architectures with pooling as first operations and with fully connected layers before convolutional layers are removed from the search space. Classical classification CNNs like VGG-16 [29] are contained in this search space.

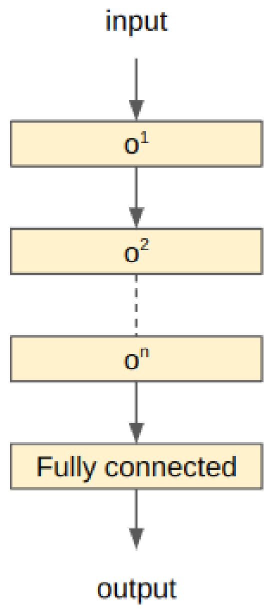

Figure 3.2: Chain-structured search space.

In another work [42], a relaxed version of the simple chain-structured search space is proposed. In this alternative, skip connections are allowed. In that case, the node $z^{(k)}$ has one default parent node, $z^{(k-1)}$, and possible ancestor nodes, depending on the architecture a sampled (equation 3-3).

$$z^{(k)} = o^{(k)}(\{z^{(k-1)}\} \cup \{z^{(i)} | a_{i,j} = 1, i < k-1\}) \tag{3-3}$$

In the case where exists a skip connection, a concatenation is performed. The addition of the skip connection made it possible to represent more complex networks including ResNet [31] and DenseNet [32].

Other works proposed search spaces [43, 44, 45] that allowed for more complex multi-branch architectures, increasing the degrees of freedom and allowing for even more complex architectures. However, having a more extensive search space increases the challenge of NAS.

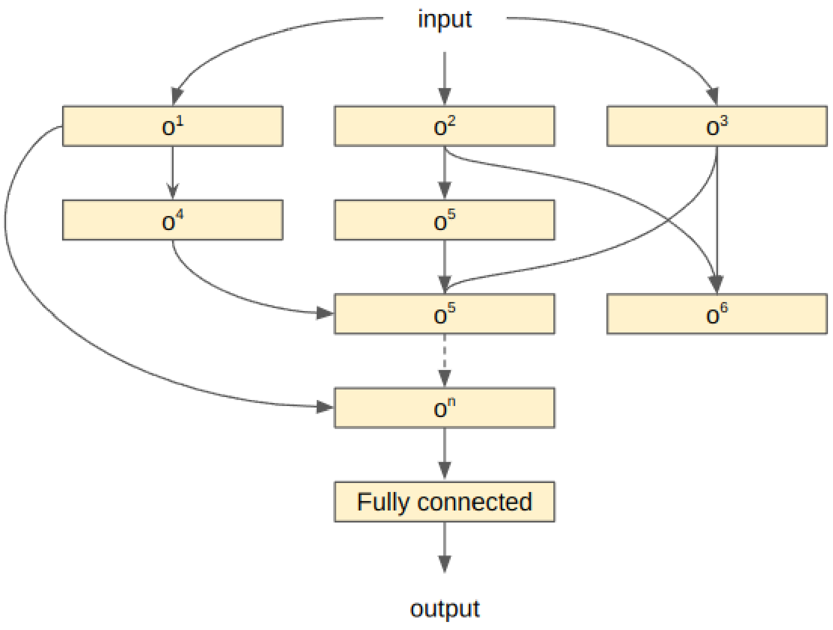

Figure 3.3: Multi-branch search space.

3.2.2 Cell Search Space

As we can see from VGG-16, Figure 2.6, shown in Chapter 2 and by other well-known architectures from the literature like ResNet, DenseNet, Inception-Net, etc., handcrafted DNNs present architectures that rely on repeating motifs or structures. These repeating structures may be referred to as cells or blocks. Cell search spaces leverage this idea of repeating patterns in the architecture to perform NAS. Accordingly to [1], this approach yields the following advantages:

- The size of the search space can be significantly reduced. Since each cell of a repeating structure network has fewer layers than the whole network.
- Architectures built from cells are more easily adapted to other datasets, either by varying the number of cell repetitions or changing the number of filters of the model.

– The literature has many examples of DNNs with repeating patterns, confirming the idea that stacking cells can be beneficial for feature extraction.

One of the earliest works to explore cell search spaces was [46]. In this work, the popular NASNet search space was proposed. This search space is based on the typical structure of a CNN, where convolutional blocks are stacked followed by pooling layers. Two repeated motifs are searched in NASNet, the *Normal Cell* and the *Reduction Cell*. Both are convolutional cells, but the normal cell produces a feature map as output, which has the same size as the input feature map, while the reduction cell produces a feature map that is reduced by a factor of 2. Each cell is a small DAG representing a feature transformation, as the search space was described at the beginning of the Chapter.

The cells are organized on a template (Figure 3.4). In this template, a repetition of N normal cells is followed by a reduction cell in a repeating way. In order to perform classification, the last layer is fully connected with softmax activation. N is a predefined hyperparameter. For the template proposed for ImageNet, a first layer 3x3 convolution is fixed and its role is to extract low-level features from the input. In the NASNet search space, the skip connections are subject to search, so each cell can take as input two previous cell outputs.

Similarly, other works [47, 48] proposed a similar search space, but searching for only one cell and fixing the reduction operations, instead of having normal cells and reduction cells.

Despite the advantages of cell search spaces, this approaches deals with NAS on the micro level (inside the cells) and the macro level is still manually defined.

3.3 Search Strategy

The objective of the search strategy is to find an architecture a from the search space A that maximizes an objective function on unseen data. This objective function may refer to a performance metric, but can also be multiobjective, e.g. maximizing performance while minimizing the number of parameters. Furthermore, the search strategy defines how architectures from the search space will be sampled to perform better on the validation set when trained on the training set (Figure 3.1). The search space is potentially exponentially large; therefore, the choice of the search strategy deals with the exploitation-exploration trade-off, since quick convergency, despite desirable, may lead to poor-performing DNNs.

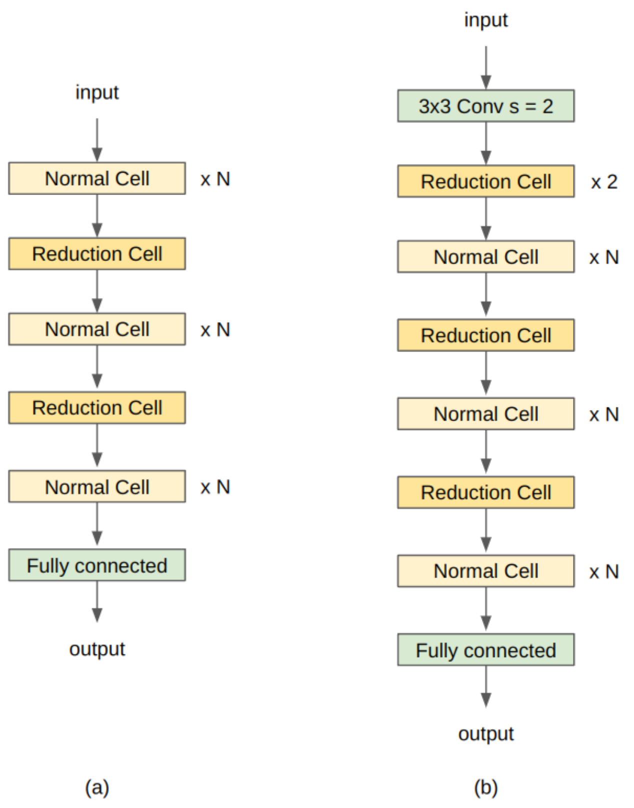

Figure 3.4: Templates proposed for the NASNet search space (a) template proposed for CIFAR10 (b) template proposed for ImageNet. The green cells are fixed cells in the template.

NAS is a search problem, and several algorithms have been used in the literature, i.e., random search, reinforcement learning, evolutionary algorithms, etc.

3.3.1 Reinforcement Learning

Reinforcement learning (RL) methods are used in order to model sequential decision-making processes. In RL methods, an agent interacts with the environment by taking actions and receiving rewards. The goal is to optimize a policy that makes the agent select its actions to maximize future returns. Return is the cumulative discount sum of rewards over multiple interactions with the environment. The policy defines how the agent will act based on the environmental state. And the environment's state is obtained by observation 3.5.

When framing NAS as an RL problem, the sampling of a candidate architecture is the agent's action, and the RL's action space is the NAS search space. The return required by the RL algorithm is the evaluation of the candidate's architecture. Different works applying RL to NAS differ

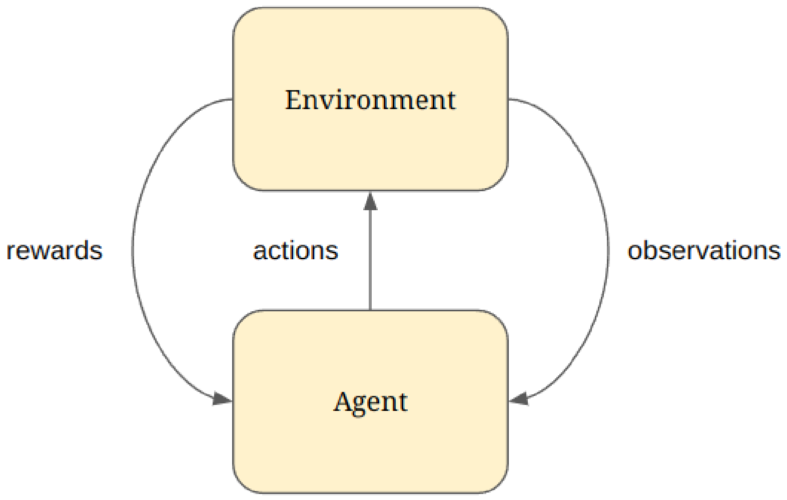

Figure 3.5: A general framework for RL methods.

in the policy representation and optimization method. [42] uses a recurrent neural network (RNN) policy and trains it with REINFORCE [49]. The RNN sequentially samples a variable-length string that is decoded into the candidate architecture. In a follow-up work [46], they used Proximal Policy Optimization instead of REINFORCE. Another alternative is used by [41]. In this work, Q-Learning trains a policy to choose a layer and its hyperparameters sequentially. Despite the layers of the architecture being sequentially sampled, the reward is only given once the whole architecture is evaluated. This approach simplifies the RL problem to a stateless multi-armed bandit problem.

A different approach frames NAS as a sequential decision-making process. In [50], the reward is the estimated performance of a partially trained network and the action corresponds to the application of a function-preserving mutation in the architecture. The policy for this work is also an RNN trained with REINFORCE.

RL-based approaches are a popular choice to tackle the NAS problem and several other works explore it. The next section covers another popular choice for the search strategy which is evolutionary algorithms.

3.3.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are population-based optimization algorithms inspired by the natural process of evolution. The algorithm comprises some key steps: initialization, parent selection, recombination and mutation, and survivor selection [51]. The first step is initialization which defines how the first population is created. After initialization, each generation is composed of the following steps:

1. Select parents from the population for reproduction.

2. Apply recombination and mutation operations to create offspring with higher variability.

3. Evaluate the fitness of the offspring.

4. Select the survivors of the population based on their fitnesses.

The generations are repeated until a stop criterion is reached, which could be a maximum number of generations or a fitness requirement. The whole process is in Figure 3.6.

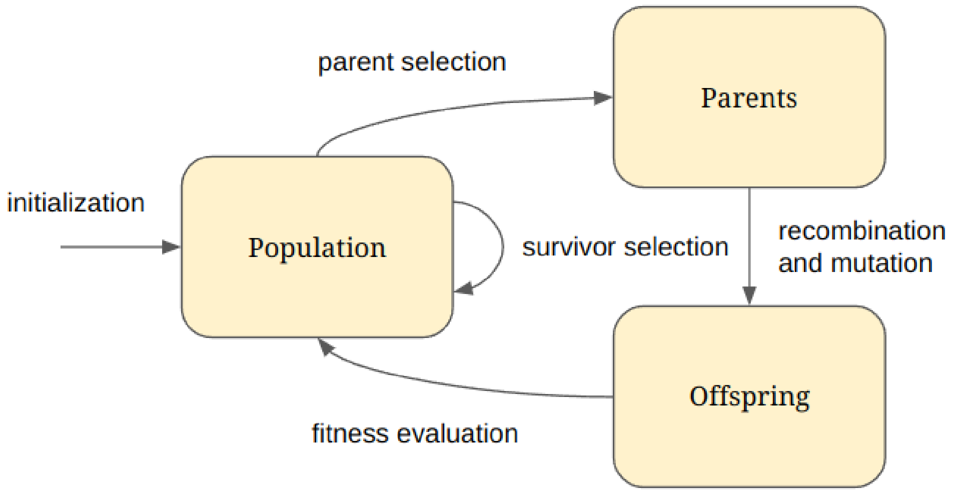

Figure 3.6: A general framework for EA methods.

When applying EAs to NAS, the population is a set of candidate architectures sampled from the search space and its fitness is the validation set's evaluation. Usually, only the mutation operation is applied for NAS, since there is no indication that combining two candidate architectures with high fitness will lead to a good architecture.

Different works using EAs on NAS differ on how they encode and represent architectures (individuals), the selection, and the update operations.

One of the first works to find competitive classification CNN architectures using EAs was [52]. In their approach, the initialization step starts with the definition of one thousand simple architectures, composed only of a global average pooling layer and the fully connected output layer. Then two architectures are sampled at random from the population for evaluation. The better performing one is copied and mutated to generate a child architecture.

Then the parent and the child architectures are added back to the population, while the worse architecture is removed. The possible mutation operations include adding or removing convolutional layers, changing hyperparameters, adding or removing skip connections, and so on.

In [43], the architecture is represented by a sequence of triplets defining each block's operation and inputs. The string formed by the concatenation of those triplets forms the genotype. Since that representation may lead to unused inputs or disconnected blocks, some parts of the genotype may be inactive and therefore, not present in the decoded architecture (phenotype). They use $(1+\lambda)$-evolutionary strategy ($\lambda = 2$) approach [53]. The mutation is generated by changing parts of the genotype and the fitness value is the accuracy metric on the validation set. The parent selection method is the tournament.

Lie et al.,[48] proposed a hierarchical search space and at every generation, a hierarchical level is selected to be mutated. The population is initialized with two hundred simple individuals, diversified by a thousand random mutations. Mutations can add, change or remove operations from the genotype on a hierarchical level. Parents are selected through a tournament and no individuals are removed from the population during evolution.

In [54] an EA was applied to the NASNet search space, leading to the discovery of AmoebaNet-B and AmoebaNet-C. The evolutionary algorithm used a tournament for parent selection, and the selected individuals are mutated with a random operation change or connection change. For survival selection, the fitness value and the age of the individual are considered. That way, individuals that lived for multiple generations tend to be removed from the population. This provides a regularization effect that adds diversity and ensures that a good performance on the validation set is the only way an architecture keeps in the population for longer. [54] also conducted a case study comparing RL and EA, concluding that EA performs as well as RL in terms of accuracy, but EA is able to find smaller architectures.

3.3.3 Other methods

The majority of works in the field of NAS use either RL or EA. However, with the growing interest in this research field, other techniques have been proposed, such as Bayesian Optimization [55], Monte Carlos Tree Search [56, 57], Hill Climbing Algorithm [58] and Gradient-based optimization [59, 60, 61, 62, 63] for a continuos-relaxed version of the search space.

3.4 Performance Estimation Strategy

As previously mentioned in the Search Strategy section, the objective of NAS is to find a neural architecture a that maximizes the performance on unseen data. This performance may be a task-specific metric, or NAS may also deal with multi-objective optimization (e.g. finding good-performing yet small architectures). The most straightforward way of assessing a candidate's architecture a performance is to train it on a training set and evaluate it on a validation set. However, this process is time and resource-consuming, sometimes achieving thousands of GPU days [46, 52, 54].

In that context, it is interesting to use methods to reduce the cost of estimating the performance of a candidate architecture. For example, some works propose lower-fidelity estimates of the performance, like training for a reduced number of epochs [64, 65], training on a subset of the training set [66], training with lower-resolution input images [67], reducing the number of filters of the convolutional layers or reducing the number of cells in the network template [46, 68]. These lower-fidelity estimation strategies effectively reduce the time and resource consumption of the search process. However, this strategy may introduce error when the ranking produced by the estimates is different from the ranking that would be achieved in case of a full evaluation.

Another approach is to extrapolate the learning curve. Several works applied this concept in different ways. [69] propose to extrapolate the learning curve to identify candidate architectures that would probably not achieve a good performance and interrupt their training and evaluation process. That would speed up the NAS process because only promising candidate architecture would be fully evaluated. Other works [68, 70, 71, 72] consider architectural hyperparameters to predict which partial learning curves are most promising. [73] trains a surrogate model to predict the performance of candidate architectures based on previously evaluated architectures using its architecture properties as predictive variables.

A third approach is to initialize the weights from a previously trained architecture. This is done by initializing the weights of candidate architectures based on the weights of previously trained ones. Network morphisms [74] is a mechanism that allows generating child architectures from parent ones preserving their function, several works already use this mechanism [50, 58, 75]. The weight inheritance allows for candidate architecture not to be trained from scratch, significantly reducing the time consumption of the evaluation.

As stated in Section 3.2, an architecture can be viewed as a DAG. For One-shot Architecture Search, the candidate architecture DAGs are subgraphs of a supergraph (the One-shot model). The subgraphs share weights and edges. Therefore, in that approach, the One-shot model is trained once before the search. During the search, the candidate architectures are sampled from the One-shot model and their weights are inherited from the One-shot model. This significantly speeds up the evaluation process since no training is required. However, it is important to notice that since all subgraphs are trained together, the performance estimation of the candidate architectures can incur in a large bias and do not correspond to the performance those architectures would achieve if trained from scratch. In addition, one limitation of One-shot methods is that the whole supergraph has to fit on memory.

Despite the great focus on the strategies to reduce time and resource consumption of the performance estimation process, noise in the estimation is also important. Estimating the performance of a DNN is a stochastic process, due to several factors like weights initialization, training, and dataset split. This issue is particularly problematic for small datasets. Dushatskiy et al.,[76] showed that performing 5-fold cross-validation with 3 different initialization leads to a more reliable performance estimation than the straightforward single initialization single evaluation. On the other hand, this increases time and resource consumption.

3.5 Neural Architecture Search applied to Semantic Segmentation

With the success of NAS on classification, other works applied to other computer vision tasks like object detection and image segmentation. This section will cover some works on NAS applied to semantic segmentation.

The first work applying NAS to semantic segmentation was done by Google DeepLab [77]. After the outstanding performance achieved by their family of proposed semantic segmentation architectures DeepLab-v1 [78], DeepLab-v2 [79] and DeepLab-v3 [80], they proposed an architecture search space to search for a dense prediction cell (DPC), that produces segmentation maps from the features extracted by a backbone. In their work, they used MobileNet-v2 [81] and Xception [82] as backbones. The DPC consisted of a DAG with n branches and each branch was specified by a 3-tuple (X_i, OP_i, Yi), where X_i is the input tensor, OP_i is the operation performed by the branch and Y_i is the output tensor. The operation could be:

- 1×1 convolution
- 3×3 atrous separable convolution with rate $r_h \times r_w$, where r_h and r_w $\in \{1, 3, 6, 9, ..., 21\}$

– Average spatial pyramid pooling with grid size $g_h \times g_w$ where g_h and g_w $\in \{1, 2, 4, 8\}$

In order to reduce time consumption, they trained the backbone on a proxy task and cached the weights, so that when evaluating a DPC, they can train for fewer epochs. The search strategy used in this work was random search [83].

Another work from Google DeepLab [56] applied NAS to the backbone of the architecture and used Atrous Spatial Pyramid Pooling (ASPP) modules to produce the segmentation maps. The search space is divided into two, the cell level search space and the network level search space. The cell is a convolutional module composed of n blocks, and each block is composed of a two-branch structure mapping 2 input tensors to 1 output tensor. Each block is specified by a 5-tuple (I_1, I_2, O_1, O_2, C), where I_1 and I_2 are the input tensors, O_1 and O_2 are the operations performed on each input and C is the operation used to combine the tensor produced by the operations. The operations belong to the set of operations:

– 3×3 depthwise-separable convolution.
– 5×5 depthwise-separable convolution.
– 3×3 atrous convolution with rate 2.
– 5×5 atrous convolution with rate 2.
– 3×3 average pooling.
– 3×3 max pooling.
– skip connection.
– no connection.

A network template was defined on the cell search spaces covered in Section 3.2. In Auto-DeepLab [56], they set a continuous relaxation approach in order to find the network-level architecture of the decoder. Furthermore, they adopt a gradient-descent approach to perform the search.

Following the idea of having a network-level search space and a cell-level search space, C2FNAS was proposed [84]. A bi-level search (coarse-to-fine) is performed, in which first the macro-level (coarse stage) search is performed and the topology of the network is defined and then the micro-level (fine stage) search is performed in order to define the operations each cell performs.

During the coarse stage, a default operation is attributed to each cell, and use an EA performs the search for topology. The topology incorporates two priors: (1) U-shaped encoder-decoder structure and (2) skip-connections between encoder and decoder. After the topology is defined, the fine stage takes place. During that stage, the topology is fixed and One-shot NAS is performed in order to define the operation in each cell.

Different search strategies are used in the coarse and fine stages because the topology search space is reduced when compared to the cell level. So an EA is applied to the small search space because it needs every candidate to be evaluated and then One-shot is applied to the big search space because, in that case, the whole search space (supergraph) is trained once.

The possible operations belong to the set:

- $3 \times 3 \times 3$ convolution.
- $3 \times 3 \times 3$ depthwise-separable convolution.
- $3 \times 3 \times 1$ convolution.

The cell is composed of either one or two branches (in case there is a skip connection). In the case of one branch, the input tensor passes through a $1 \times 1 \times 1$ convolution, then the cell operation is applied, and finally, a $1 \times 1 \times 1$ convolution is applied. In the case of a two branches cell (with skip-connection), both branches go through a $1 \times 1 \times 1$ convolution and cell operation. Finally, the output tensor is summed and passes through a $1 \times 1 \times 1$ convolution again.

MixedBlock [85] used a similar search space as CF2NAS [84] composed by a topology-level and a cell-level search space. However, this work presents two main differences: (1) the topology and cell search were performed simultaneously, and (2) instead of defining cells that perform simple operations, convolutional blocks used by state-of-the-art classification CNNs were used. So the operation performed by each cell could be one of the following:

- VGG block [29].
- ResNet block [31].
- DenseNet block [32].
- InceptionNet block [30].

In this work, Local Search was used as the search strategy.

Both CF2NAS [84] and MixedBlock [85] search spaces produced networks with great inspiration from encoder-decoder architectures like U-Net. Another work that took inspiration from the U-Net architecture, but using cell search spaces, was NAS-Unet[86]. In this work, two types of cells were defined: (1) the DownSC cell that halves the resolution of the feature maps and (2) the UpSC cell that doubles the resolution of the feature maps. The U-Net architecture is used as a network template; the encoder is composed of DownSC cells and the decoder of UpSC cells. Each cell is a DAG that receives possibly 2 inputs (the previous cell output and skips connection when applied) and produces one output. The number of operations inside the DAG is parametrizable. DownSC cells have only downscaling and normal operations, while UpSC cells have upscaling and normal operations.

4
Q-NAS

This chapter presents Q-NAS (Quantum Inspired Neural Architecture Search). In the first section, we overview the algorithm; then, we present the search space and how Q-NAS represents the individual. Finally, we will go through the quantum-inspired evolutionary algorithm used in Q-NAS.

4.1
Overview

Q-NAS is a quantum-inspired evolutionary algorithm (QIEA) developed to perform NAS. Q-NAS searches for the best DNN architecture and hyperparameters for a predefined task. When proposed, Q-NAS was applied to perform image classification [3]. However, it's core QIEA was designed to be task agnostic. So, it could potentially be extended to any task.

Figure 4.1 shows an overview of Q-NAS. Before running Q-NAS, the user must predefine the network template, function set, hyperparameters, and their ranges. This information characterizes the search space and is closely related to the specific task. In Figure 4.1, we see a network template, where the network is represented by a chain structure of nodes followed by a fully connected layer to produce the output. The function set comprises all possible operations a node in the network template can perform. So one can define a function as a simple layer or a complex block.

With that defined, Q-NAS is run, and the output is produced. The output of Q-NAS is the best architecture found in the search process and its hyperparameters.

In order to address the search for network architecture and hyperparameters, the quantum chromosome is divided into two parts. The numerical part accounts for the hyperparameters, and the categorical part accounts for encoding the functions each node from the network performs.

Figure 4.2 shows an overview flowchart of QNAS. During initialization, the predefinitions are made, the generation count is set to 0 and the quantum population is initialized. The quantum individual can not be directly evaluated, so in each generation, the quantum population is observed in order to generate the classical population. Then, the classical population is evaluated and the

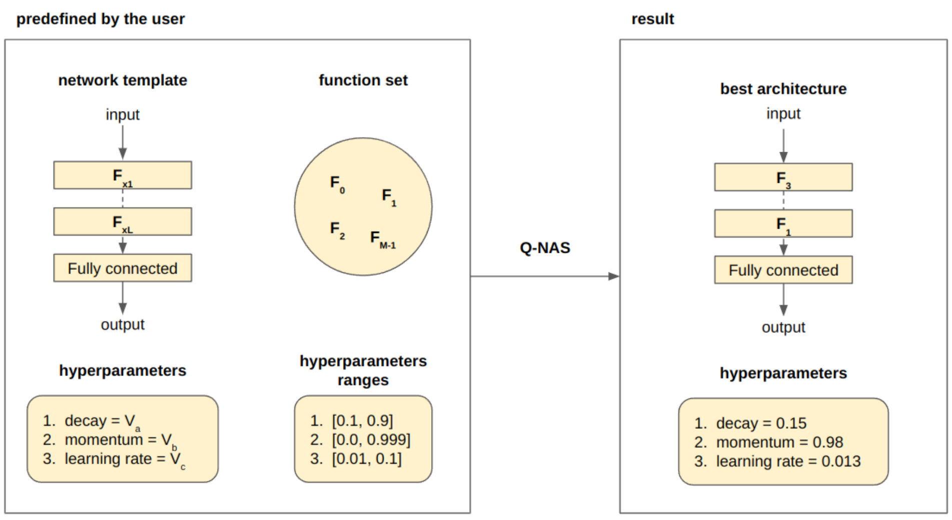

Figure 4.1: Q-NAS overview. The user must predefine the network template, function set, hyperparameters, and ranges. Q-NAS produces as output the best architecture and its hyperparameters.

fitness of the individuals is used to update the quantum individuals. The generation of the classical individual and the update of the quantum individual is done differently for the categorical and numerical part of the chromosome. The process ends when the maximum number of generations is reached.

4.2 Search Space and Chromosome Representation

The quantum individual representation is a key concept for quantum-inspired evolutionary algorithms since quantum individuals represent many quantum states, and when observed, the classical individual can be decoded into a possible solution to the problem. As stated before, Q-NAS quantum individual is two-fold: the numerical part and the categorical part.

4.2.1 Numerical quantum chromosome

Given G hyperparameters the quantum chromosome q_i is defined as:

$$q_i = [p_{i1}(x), ..., p_{iG}(x)] \tag{4-1}$$

Where each $p_{ij}(x)$ is a probability density function (PDF) that represents the probability of observing a specific range of values for the hyperparameter j when the individual i is observed. Q-NAS defines the $p_{ij}(x)$ as square pulses that are initially defined by their predefined ranges (l_{ij} for the lower bound and u_{ij} for the upper bound).

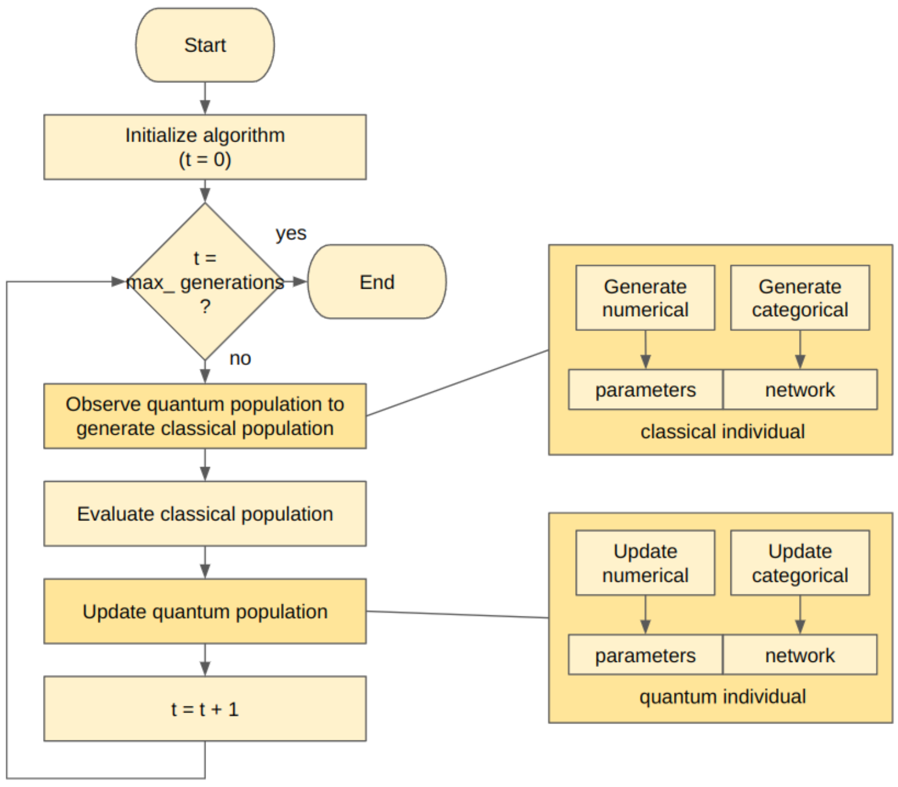

Figure 4.2: Q-NAS simplified flowchart showing in the dark boxes the steps are executed differently for the two parts of the chromosome.

The *generate numerical* step in Figure 4.2 represents the observation process. The observation of $p_{ij}(x)$ follows the steps:

1. generate a random number r in the interval $[0, 1]$.
2. find x such that $F_{ij}(x) = r$, where $F_{ij}(x)$ is the cumulative density function (CDF) of $p_{ij}(x)$
3. the observed hyperparameter value is $x = r \times (u_{ij} - l_{ij}) + l_{ij}$

After observation, the numerical part of the classical individual is generated. A crossover operation is then applied to the classical individual.

Another step that has to be defined is the quantum individual update. A random mask is generated based on the parameter *update_quantum_rate* to define, which $p_{ij}(x)$ are going to be updated. The idea is that $p_{ij}(x)$ is changed in order to increase the probability of observing values that lead to better-evaluated individuals. This is done by changing the mean μ_{ij} and the width σ_{ij} of the $p_{ij}(x)$ using the following equation (Eq. 4-2).

$$
\begin{aligned}
d^t &= \max_{i=1..N} c^t_{ij} - \min_{i=1..N} c^t_{ij} \\
\mu^{t+1}_{ij} &= \mu^t_{ij} + r \times (c^t_{ij} - \mu^t_{ij}) \\
\sigma^{t+1}_{ij} &= \sigma^t_{ij} + r \times (d^t - \sigma^t_{ij})
\end{aligned}
\tag{4-2}
$$

Where c^t_{ij} is the jth current value of a classical individual i and r is a random number in a range $[0, 1]$. Rewriting the Equation 4-2 in terms of l_{ij} and u_{ij} we have:

$$
\begin{aligned}
l^{t+1}_{ij} &= l^t_{ij} + r \times \left(c^t_{ij} - l^t_{ij} - \frac{d^t}{2} \right) \\
u^{t+1}_{ij} &= u^t_{ij} + r \times \left(c^t_{ij} - u^t_{ij} + \frac{d^t}{2} \right)
\end{aligned}
\tag{4-3}
$$

The Equation 4-3 can be interpreted as the PDF defined by $p_{ij}(x)$, being shifted and narrowed in the direction of the value of the best classical individual hyperparameter.

4.2.2 Categorical quantum chromosome

For the numerical part of the quantum chromosome, the observation already produces the values of the hyperparameters that are going to be used. On the other hand, for the categorical part of the chromosome, the observation process will output the functions from the function set that each node is going to perform. A decoding process must take place to translate these into the network itself, using the network template so the individual can be evaluated.

The network template proposed in [3] is a chain-like structure composed of L nodes, and every node performs a function F from the function set. As it was proposed to be applied to image classification, a final fully connected layer is fixed.

As shown in Figure 4.1, the user predefines M functions. The functions are mapped to integers in the $[0, M-1]$ range. That way the classical individual p_i is defined as:

$$
p_i = [g_{i1}, .., g_{iL}], g_{ij} \in [0, M-1]
\tag{4-4}
$$

Considering this classical individual representation, the quantum individual defines a probability mass function (PMF) for each node in the network template. A quantum gene encoding a node is then represented by:

$$g_j = [x_{j1}, .., x_{jL}], x_{ik} \in [0, 1) \text{and} \sum_{k=1}^{M} x_{jk} = 1 \tag{4-5}$$

where x_{jk} is the probability the function k is observed in the classical individual for the node j.

That way, if we have N individuals, the quantum population can be represented by an array of shape (N, L, M).

At the initialization step, every gene g_j starts with the same PMF. The user can also specify this so that an initial bias towards some functions can be inserted. After that, the observation process is simply the use of the probabilities x_j to sample which function is going to be observed.

Figure 4.3 shows the process of observing a quantum individual, generating a classical individual, and then decoding it into the network to be evaluated using the network template. As an example, the quantum gene g_0 is composed by a set of probabilities $[0.2, 0.1, 0.3, 0.1, 0.3]$. Each probability value in the vector g_0 defines the sampling probabilities of the associated functions. So, for the case shown in Figure 4.3, the function 0 was sampled, and it had 20% probability of being observed. The function 0 in the function set is the 1×1 convolution with 32 kernels.

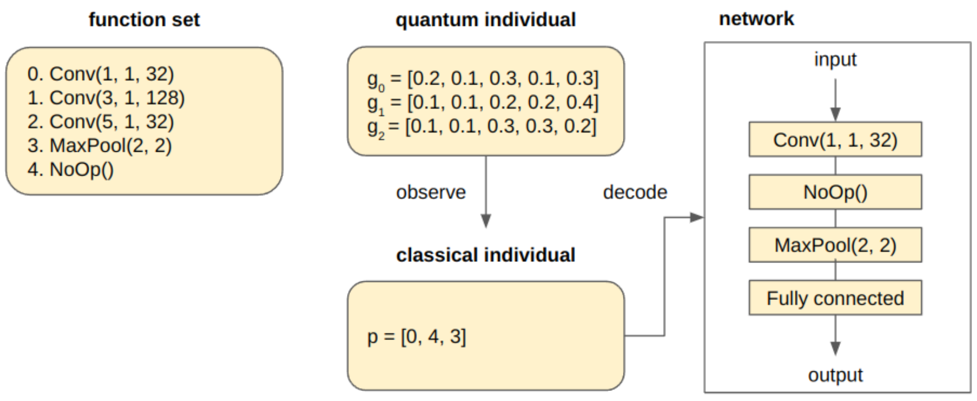

Figure 4.3: The observation process generates a classical individual from the quantum individual. The classical individual is then decoded into the network to be evaluated.

The classical individual can lead to unfeasible networks. For example, for a given input image resolution, there is a maximum number of pooling operations allowed. In Q-NAS, a *penalize_number* parameter is defined so that only that number of pooling layers is decoded. The exceeding layers are ignored during the decoding process. However, a decrease of 0.01 for each exceeding pooling layer is applied to the fitness value of that individual.

Following the same idea that the best classical individuals should guide the update of the quantum chromosome, the update of the categorical quantum chromosome aims to increase the probability that functions observed in a good-performing classical individual should be increased while the remaining decrease. Q-NAS uses a simple heuristic for that. First, a random mask, based on the *update_quantum_rate* parameter is generated to update the quantum individual. This mask defines which g_i are going to be updated, then for each node i in the mask the following steps are executed:

1. get the function for node i in the best classical individual
2. calculate $update\_value = r \times 0.05$ where r is a random number in the interval $[0, 1]$
3. increase the probability of observing that function by *update_value*
4. decrease the probability of observing the other functions by $\frac{update\_value}{M-1}$

These steps ensure that the probability of observing any function will never be 0.

4.3 Quantum Inspired Evolutionary Algorithm Search Strategy and Evaluation Strategy

After defining the Search Space, how the quantum individual is represented and evaluated. This section covers the steps of the quantum-inspired evolutionary algorithm and evaluation strategy used in [3].

The algorithm of Q-NAS is stated below (Algorithm 1).

First, the quantum population $Q(t)$ is initialized. $Q(t)$ is composed of N quantum individuals $q_i^t, i = 1, 2, .., N$, where the size of the quantum population is defined by the *number_of_individuals* parameter. The initialization process involves assigning the initial probabilities to the quantum individuals. For the numerical part, the user provides lower l_{ij} and upper u_{ij} bounds of the PDFs, and for the categorical part, the user provides the initial probabilities of observing each function of the function set. If not provided, the probabilities will be assumed the same.

Then the classical population is generated by observing both parts of the quantum individuals' chromosomes. These procedures were already discussed. One important aspect is that each quantum individual can generate one or more classical individuals. Because of the stochastic nature of the observation process, each observation of the same quantum individual may lead to different

Algorithm 1 Q-NAS

```
$t \leftarrow 0$
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 $C(t)$
        $P(t) \leftarrow$ best individuals from $[C(t) \cup P(t)]$
    **end if**
    $Q(t+1) \leftarrow$ update $Q(t)$ based on $P(t)$ values
    $t \leftarrow t + 1$
**end while**
```

classical individuals. The number of classical individuals generated for each quantum individual is defined by the parameter *repetition*.

After that, each classical individual is decoded and evaluated. The evaluation process follows the steps:

1. train the decoded network for 50 epochs on a subset of the training set.
2. evaluate the accuracy of the network on the validation set for the 5 last epochs.
3. assign the best of the 5 accuracies as the individual fitness.

Training for a restricted number of epochs and on a subset of the training set are some strategies used to reduce the time consumption of Q-NAS. Another restriction imposed is that networks that took more than 90 minutes to train are evaluated with a fitness value equal to 0.

In the first generation ($t = 0$), the best population $P(t)$ will be $C(t)$. For the other generations ($t \geq 0$), there is already a previous $P(t)$, so recombination can be applied. Crossover is applied to the numerical part of the chromosome and no recombination is applied to the categorical part, since there is no indication that this can lead to better network architectures [2]. Q-NAS uses the steady-state approach for selecting $P(t)$. That means that if the classical population has size K ($K = N \times$ *repetition*), $P(t)$ will save the K best individuals from the set $P(t) \cup Q(t)$.

Finally, quantum individuals are updated based on $P(t)$ following the procedures previously defined, based on the parameter *update_quantum_gen*, which defines the periodicity of the quantum individuals' update. Therefore, for example, if $update\_quantum\_gen = 3$, the quantum individuals are going to be updated every 3 generations.

The algorithm is run for $T = max\_generations$. After its completion, the best architecture is trained for 300 epochs using the best hyperparameters on the whole dataset.

5
SegQNAS

In this chapter, we present our method, SegQNAS, which applies Q-NAS to search for semantic segmentation neural networks.

In order to extend Q-NAS to perform NAS for semantic segmentation networks, an adequate search space must be used. This is done by changing the network template and function set predefinitions and adjusting the performance estimation strategy. The decoding process had also to be changed to account for the design constraints imposed on the network topology. The Quantum Inspired Evolutionary Algorithm, the core of Q-NAS, is task agnostic. Therefore the search strategy remains the same.

SegQNAS was implemented using the Keras framework [87].

One observation is that SegQNAS focused on the evolution of the network architecture; thus, the hyperparameter search is not covered.

5.1 Search Space

The search space proposed for SegQNAS is based on the works of [84, 85] that divide the search space into the topology and cell levels. The following sections will cover this two level search space.

5.1.1 Topology level search space

The topology level search space is shown in Figure 5.1. The network is represented by a DAG composed of L cells. Each cell is on a level in the $[0, D]$ range. D defines the maximum allowed level. At the level d, the feature maps have shape $\frac{R}{d^2}, \frac{R}{d^2}, F \times d^2$, where (R, R) is the resolution of the input image and F is the number of kernels (number of filters) of the stem convolution. The cell level is subject to four design constraints:

1. The cell level either increases by 1 unit, decreases by 1 unit, or remains the same as the previous cell.

2. The output cell level is the same as the input cell level.

3. The minimum cell level is 0.

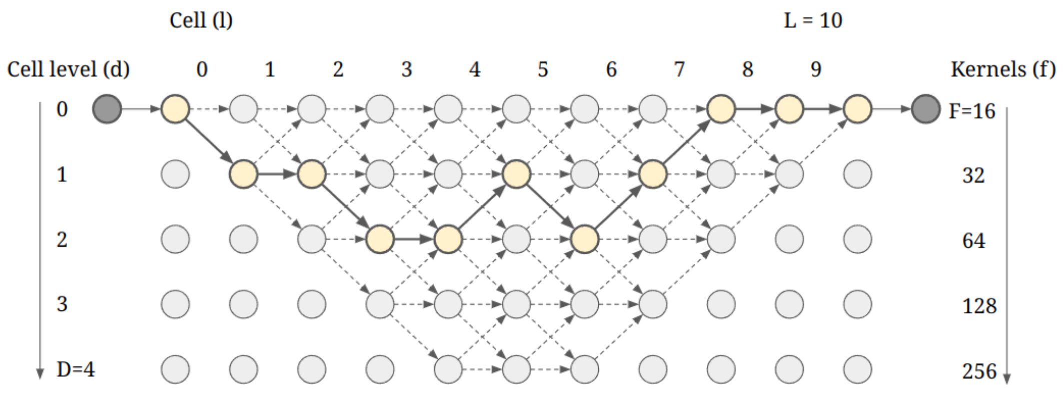

Figure 5.1: One possible network in the topology search space. The circles represent feature maps, while the arrows represent the cells. The dark grey circles represent the network's input (left) and the output (right). The yellow circles are feature maps that are outputs of the network's cells. The first and the last arrows represent, respectively, a stem convolution to adequate the feature map shape and the final convolution that produces the segmentation maps. The solid arrows are cells of the network, and the dashed arrows represent other possibilities in the search space; therefore, feature maps that are not connected by any arrow are infeasible. Skip connections are omitted for illustration purposes. The cell level is on the right ranging from 0 to 4 ($D = 4$), and on the top, the cell enumeration ranges from 0 to 9 ($L = 10$).

4. The maximum cell level is D.

The first constraint ensures that the features are extracted and recovered gradually by the network. The second constraint is a requirement for segmentation networks since it is needed that the dimensions of the output match the dimensions of the input. The third and fourth constraints are just limiters of the search space.

With that in mind, each cell could be of one of the types: upscaling, downscaling, or nonscaling. The upscaling cell increases the cell level by 1 unit, causing the feature map resolution to double while halving the number of channels. The downscaling cell halves the feature map resolution while doubling the number o channels, while the nonscaling cell preserves the same feature map shape as the previous cell. That relationship between the feature map resolution and the number of kernels with the cell level is expressed by the resolution of the feature map at the cell level d being $\frac{R}{d^2}, \frac{R}{d^2}, F \times d^2$ where (R, R) is the resolution of the input image and F is the number of kernels (number of filters) of the stem convolution.

In order to enforce the aforementioned design constraints and prevent infeasible solutions from being decoded, a special decoding process is proposed and will be described in Section 5.2.

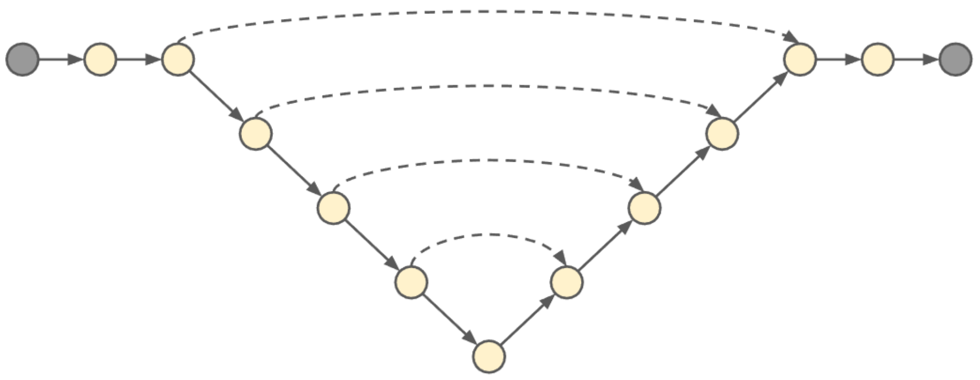

Figure 5.2: U-Net topology represented in SegQNAS search space. The dashed arrows connecting the encoder and decoder are the skip-connections.

Another design choice for the search space was the addition of skip-connections during the decoding process. The skip-connections are added when there is a a previous cell on the same level. This is similar to the proposed skip connections in the U-Net architecture [36].

The proposed topology search space allows for a variety of topologies while still having as subset encoder-decoder topologies like U-Net (Figure 5.2) and its variants.

5.1.2 Cell level search space

SegQNAS search space is a global search space, in which each cell performs a different operation. That means that we are not searching repeated cells. Instead, following the strategy in [85] each cell operates a state-of-the-art CNN block. This decision allows for great flexibility, allowing each cell to perform a different operation while preventing the search space from growing too much using blocks.

The cell structure is the same in [76] (Figure 5.3). The cell is composed of a searchable block, an optional upscaling/downscaling operation depending on the cell type, and a concatenation of the skip connection and input if the skip connection is present. The input of the cell l is the output of the cell $l-1$ or the output of the stem convolution if $l = 0$.

The *concat* block in Figure 5.3 concatenates the input with the skip connection and applies a 1×1 convolution to keep the feature map shape equal to the input shape.

The upscaling operation is a 3×3 transpose convolution with $stride = 2$, and the downscaling operation is a 3×3 convolution with $stride = 2$. The number of kernels f of those operations is defined by the cell level $f = F \times d^2$, where F is the number of kernels for the stem convolution and d the level.

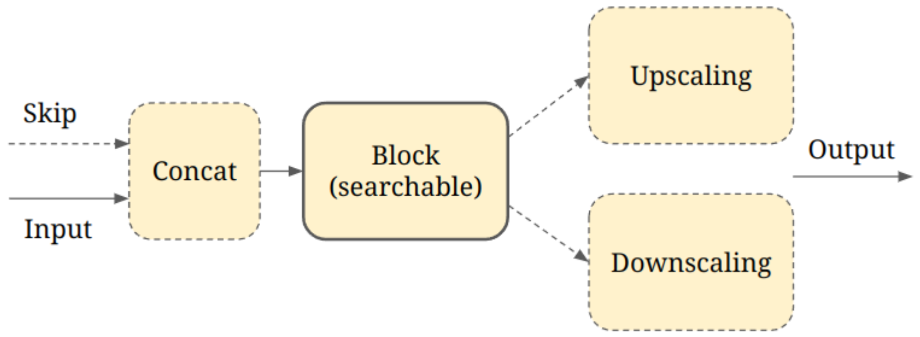

Figure 5.3: Cell structure. The dashed blocks exist only when applicable. The *concat* block exists only when there is a skip connection to that cell. The upscaling/downscaling blocks exist depending on the cell type.

The stem convolution is a 3×3 convolution with F kernels fixed as the first operation of the network and used to transform the (R, R, c) input image into a (R, R, F) feature map, where c is the number of channels of the input.

The final convolution is another fixed operation but fixed as the final operation of the network, and it is responsible for getting a (R, R, K) feature map as input and transforming it to the (R, R, N) segmentation maps, where N is the number of classes of the dataset. The final convolution is a 3×3 convolution with N kernels. If $N = 1$ the activation function of the final convolution is defined as a sigmoid. If $N > 1$, the activation function is defined as softmax.

A block of Batch Normalization and ReLU activation is implicit for all convolutional operations.

The searchable block is a block belonging to the function set of SegQNAS. The selected blocks are the same as in [85]; their layers are shown in Figure 5.4 and listed below.

- VGG Block
- ResNet Block
- DenseNet Block
- InceptionNet Block
- Identity Block

Due to their good performance in classification, these blocks are also used in semantic segmentation architectures. For example, the VGG block is the standard block of U-Net and the ResNet block is the standard block of ResU-Net. An Identity block was included to provide flexibility in the architectures. So, the Identity block does not operate and the goal of adding this block is to allow for fewer parameters networks to be found.

Figure 5.5 shows a possible network from SegQNAS search space.

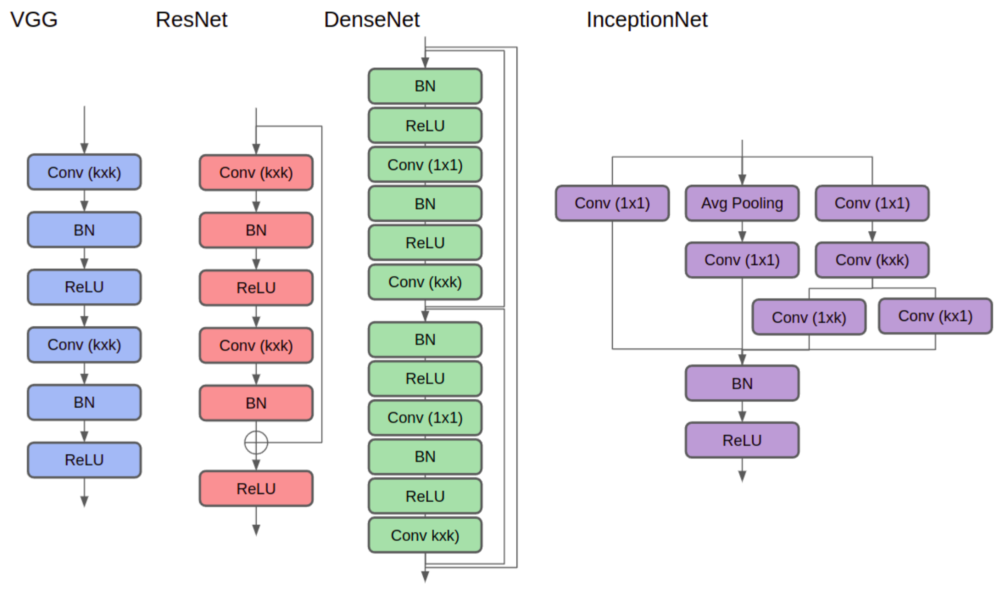

Figure 5.4: Blocks of SegQNAS. Conv(kxk) refers to a $k \times k$ convolution where k is also searchable. BN stands for Batch Normalization and ReLU indicates all activations as ReLU. All blocks produce as output a feature map with the same shape as the input.

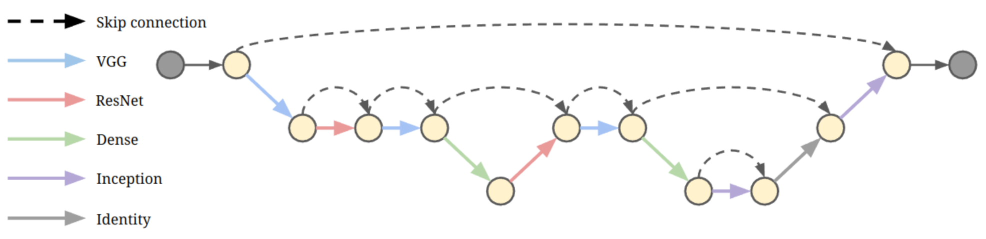

Figure 5.5: Network from SegQNAS search space.

5.1.3 Network template and Function set

To summarize the search space, each cell can be defined by three parameters:

1. The cell type (upscaling, downscaling, nonscaling).
2. The block (VGG, ResNet, DenseNet, InceptionNet, Identity).
3. The kernel size ($k \in \{3, 5, 7\}$) of the block convolutional operations.

The cell type defines the topology level search space and the block and kernel size define the cell level search space.

Therefore, the network template of SegQNAS is the DAG composed of L cells and built as shown in Figure 5.1 with the stem and final convolutions fixed.

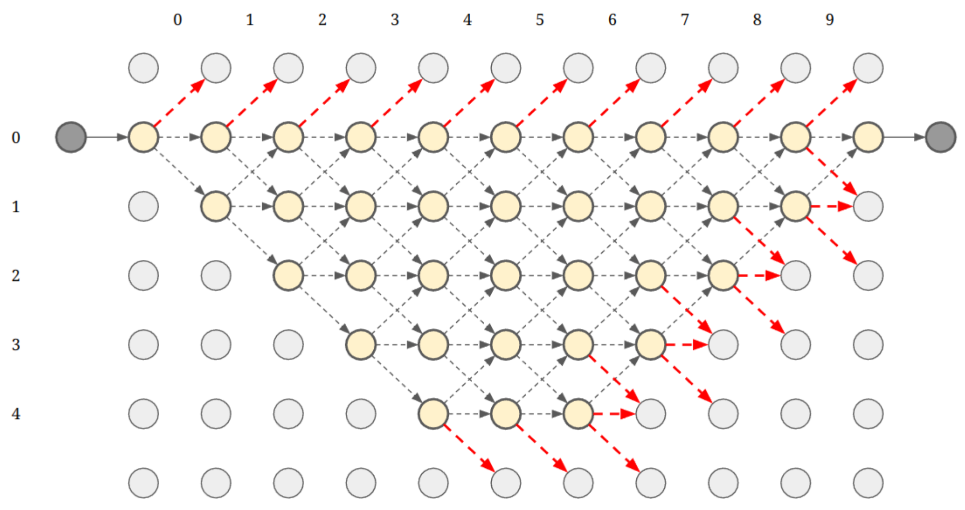

Figure 5.6: Operations that lead to infeasible networks in red. Those operations disrespect the design constraints listed in Section 5.1.

The function set of SegQNAS is the set of all combinations of cell types, blocks and kernel sizes. A caveat is that the Identity block does not need a kernel size specification as it performs no convolutional operation. So, just to give a small example, the function set could contain the following functions:

- vgg_u_3: VGG upscaling 3×3
- res_d_5: ResNet downscaling 5×5
- vgg_n_7: Dense nonscaling 7×7
- ide_u: Identity Upscaling

As there are three possible cell types, five possible blocks, three possible kernel sizes, and having in mind that the Identity block does not need a kernel size specification, we could have a function set of size $3 \times (4 \times 3 + 1) = 39$

5.2 Decoding

The decoding process that translates the classical individual into the network, using the network template, has to account for the design constraints mentioned in Section 5.1.

If we do not enforce those constraints, the decoded network could produce a segmentation map output that is not the same size as the input, with cells level above 0 or below D. Figure 5.6 shows all possible operations that lead to infeasible networks considering the network template with $D = 4$ and $L = 10$.

In order to guarantee that the decoded network is going to respect the design constraints, we opted for the approach of fixing the cell type during decoding. For that, we defined four rules:

- Rule 1: if $d = 0$ and the cell type is *upscaling*, change the cell type to *nonscaling*.
- Rule 2: if $d = D$ and the cell type is *downscaling*, change the cell type to *nonscaling*.
- Rule 3: if $L - l = d$ and the cell type is either *downscaling* or *nonscaling* change the cell type to *upscaling*.
- Rule 4: if $L - l = d + 1$ and the cell type is either *downscaling* change the cell type to *nonscaling*.

Where d is the cell level.

The rule 1 account for the constraint of not allowing levels below zero, rule 2 enforces maximum level constraint, while rule 3 and 4 guarantee that the output level is the same as the input ($d = 0$).

5.3 Performance Estimation Strategy

The quantum individual is observed, and the classical individual is generated. Then, the classical individual is decoded into the network. The next step is to evaluate this network to measure a fitness value that SegQNAS uses to continue the evolutionary process.

This fitness value must be a task-specific metric measuring the performance of the network on unseen data. When Q-NAS was proposed for deep learning, it was applied to classification tasks and it used accuracy as a performance evaluation metric. However, for semantic segmentation tasks, more appropriate metrics should be used. SegQNAS uses the Dice-Sørensen coefficient (DSC) which is defined by:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|} \tag{5-1}$$

Where X is the output segmentation mask produced by the candidate network and Y is the ground truth.

The training protocol is the same as [85]. ADAM optimizer is used with a learning rate 10^-3 and polynomial decay 0.9. The loss function is Soft Dice, a differentiable version of the DSC that uses the segmentation maps that the network outputs instead of the segmentation masks that the network outputs. A data augmentation protocol was applied with horizontal flipping, shifting, scaling, rotating, and random brightness operations.

A cross-validation approach was used because [76] showed that the stochastic nature of the neural network and the dataset split could insert noise into the evaluation process. This noise is mitigated by performing cross-validation. Thus, the evaluation process consisted of performing a 5-fold cross-validation process. The DSC on the validation set for the last 20% of the epochs of the 5 folds were then averaged. Finally, this average was taken as the fitness value of the individual that generated the network.

This process was proposed this way, due to resource constraints, and some techniques were employed to reduce time consumption in the performance estimation process. First, the training was performed for a reduced number of epochs. So instead of training the network until convergence, the network was trained for only 30 epochs. That means the DSC of the last 6 epochs of each fold were averaged to get the fitness value.

Furthermore, during the training, only half of the training set is sampled randomly and used in each epoch, expecting each image to be used at least once along the 30 epochs.

After the best architecture is found, a 3 initialization cross-validation with 5-fold is performed, and each network is trained for 100 epochs using the whole dataset. The best architecture's performance is the DSC's average over the last 20% epochs (20 epochs) for every epoch.

6
Experiments and Results

In this chapter, we present the experiments conducted in order to evaluate SegQNAS. First, we describe the datasets used in the experiments, and then the setup is presented. Next, we overview the experiment protocols, and finally present and discuss the results on each dataset.

6.1
Datasets

This section describes the datasets used to perform the experiments. Both are datasets from the Medical Segmentation Decathlon challenge [4] and are publicly available. The datasets were also used in the MixedBlock paper [85], and the preprocessing transformations were the same aiming for a more accurate comparison.

6.1.1
Spleen Dataset

The Spleen dataset contains 61 3D-CT scans from patients undergoing chemotherapy treatment for liver metastases. The target segmentation was the Spleen (single-class). This dataset was selected for the challenge due to its large variation in the field of view.

From the 61 3D-CT scans, the ground truth was provided for 41 scans, corresponding to 3650 2D slices.

As a preprocessing step, the slices were clipped between -57 and 164 Hounsfield Units (HU) and normalised using min-max normalisation.

6.1.2
Prostate Dataset

The Prostate dataset contains 48 prostate multiparametric MRI (mp-MRI) scans with target segmentation regions being the prostate peripheral zone (PZ) and transition zone (TZ). This dataset was selected for the challenge because it is a segmentation of two adjoined regions with very large subject variability.

From the 48 mp-MRI scans, the ground truth was provided for 32 scans, corresponding to 602 2D slices.

As a preprocessing step, the slices were Z-normalised.

6.2 Setup

The experiments were run on a GPU cluster with three A30 24Gb GPUs.

6.3 Experiments

The goal of the experiments is to evaluate the proposed adaptation of Q-NAS to perform semantic segmentation, namely SegQNAS. To do that, four experiments were designed with different function set predefinitions. The different function sets allow evaluation of the search space on the different levels. The goals of the experiments are listed below

- Experiment 1: Evaluate the search considering only the topology level search space.
- Experiment 2: Evaluate the search considering only the cell level search space.
- Experiment 3: Evaluate performing the search on a bi-level approach like in [84], which means performing the search sequentially on the topology level and then on the search level.
- Experiment 4: Evaluate performing the search on the topology and cell levels simultaneously.

Finally, the best architecture found for each experiment is compared, in terms of performance and number of parameters, to two state-of-the-art baselines, namely U-Net and ResU-Net, and also to the architectures found in [85] which use a similar search space, datasets, and input image resolution.

For the experiments we set $L = 10$, $D = 4$, $R = 128$, $F = 16$ $number\_of\_individuals = 9$, $repetitions = 1$, $max\_generations = 100$, $update\_quantum\_rate = 0.9$, $updated\_quantum\_gen = 1$, and equal initial observation probability for every function in the function set.

The values of the architecture parameters (L, D, R, and F) were chosen to be compatible with the work we are comparing the results.

The next sections will detail the experiments further, and present and discuss the results on both datasets.

6.3.1 Experiment 1 - Topology Search

The topology level search space is defined by the cell type. The cell type can be chosen from the set: upscaling, downscaling, and nonscaling. So we can restrict the cell block and kernel size in order to perform the search only by exploring the topology level search space. We defined the default cell block as VGG and the kernel size 3 since this is the standard U-Net block and a straightforward choice.

So, the function set is

- vgg_u_3: VGG upscaling 3×3
- vgg_d_3: VGG downscaling 3×3
- vgg_n_3: VGG nonscaling 3×3

6.3.2 Experiment 2 - Cell Search

The cell level search space is defined by the combinations of possible cell blocks and kernel sizes. The cell block is one of the sets: VGG, ResNet, DenseNet, InceptionNet, and Identity. Moreover, the kernel size is $k \in 3, 5, 7$. For experiment 2 we would like to fix the topology. For that, during decoding, the cell types of a U-Net architecture are enforced (Figure 5.2).

The function set is:

- vgg_3: VGG 3×3
- vgg_5: VGG 5×5
- vgg_7: VGG 7×7
- res_3: ResNet 3×3
- res_5: ResNet 5×5
- res_7: ResNet 7×7
- den_3: DenseNet 3×3
- den_5: DenseNet 5×5
- den_7: DenseNet 7×7
- inc_3: InceptionNet 3×3
- inc_5: InceptionNet 5×5
- inc_7: InceptionNet 7×7
- ide_7: Identity

6.3.3 Experiment 3 - Bi-level Search

Experiment 3 aims to perform a bi-level search. That is, searching first on the topology search space, and then, performing the search on the cell search space, but fixing the topology to the one previously found. Since the topology search is already performed in Experiment 1, the topology for the cell level search is fixed to the topology found in Experiment 1.

6.3.4 Experiment 4 - Full Search

For the full search, the search was performed simultaneously on the topology level and cell level search space, which means that no constraint was imposed on the function set.

Thus, in this experiment, a function set containing all combinations of cell types, blocks, and kernel sizes is used.

Table 6.1 summarizes all the experiments.

Table 6.1: Summary of the experiments

Exp.	Description	Types	Blocks	k	Search Space
1	Topology search	Nonscaling Downscaling Upscaling	VGG	3	5.9×10^4
2	Cell search	U-Net	VGG ResNet DenseNet InceptionNet Identity	3, 5, 7	1.4×10^{11}
3	Bi-level search	Experiment 1	VGG ResNet DenseNet InceptionNet Identity	3, 5, 7	1.4×10^{11}*
4	Full search	Nonscaling Downscaling Upscaling	VGG ResNet DenseNet InceptionNet Identity	3, 5, 7	8.1×10^{15}

*The experiment 3 search space size is taking into account only the cell search step.

6.4 Results and discussions

This section presents the results found for the two datasets.

6.4.1 Spleen Dataset

The results of the experiments on the spleen dataset are summarized in Table 6.2. As can be seen, the DSC obtained in all experiments was higher than both baselines, U-Net and ResU-Net. The best result was found in experiment 3, in which the bi-level search approach was performed.

Table 6.2: Spleen Dataset Experiments Results

Experiment	**DSC**	**#Params**	**GPU days**
1	0.9566 ± 0.005	$\mathbf{0.9 \times 10^6}$	**4.05**
2	0.9560 ± 0.007	5.3×10^6	4.15
3	$\mathbf{0.9583 \pm 0.006}$	2.0×10^6	4.23
4	0.9582 ± 0.008	1.0×10^6	4.29
U-Net	0.9518 ± 0.008	2.7×10^6	NA
ResU-Net	0.9525 ± 0.008	2.8×10^6	NA

It is possible to notice also that for experiments 1, 3, and 4 SegQNAS was able to find architectures with fewer parameters than the baselines. The exception was experiment 2, and that is because the topology was fixed to the same as the baseline and the function set contained more complex blocks and larger kernel sizes than the baselines.

Despite the best solution being found by experiment 3. Experiment 4 finds an architecture that presents almost the same DSC but with half of the parameters.

Table 6.3 contains the comparison for each experiment with MixedBlocks [85] results.

We notice that in experiment 1, SegQNAS was able to find an architecture that achieved the same DSC performance with fewer parameters. However, in experiment 2, MixedBlock achieved better performance and fewer parameters. However, in experiment 3, SegQNAS achieved better DSC and a fewer parameter network. Finally, in experiment 4, MixedBlock found a better-performing network, but with more than 20 times parameters in comparison to SegQNAS.

The architecture found by SegQNAS in experiment 1 is shown in Figure 6.1. It is possible to notice that when comparing to U-Nets topology (Figure 5.2), the architecture found in experiment 1 does not reach the maximum

Table 6.3: Spleen Dataset Experiments Results comparison with MixedBlock [85]

	SegQNAS		**MixedBlock** [85]	
Experiment	**DSC**	**#Params**	**DSC**	**#Params**
1	$\mathbf{0.9566 \pm 0.005}$	$\mathbf{0.9 \times 10^6}$	$\mathbf{0.9566 \pm 0.001}$	3.4×10^6
2	0.9560 ± 0.007	5.3×10^6	$\mathbf{0.9567 \pm 0.002}$	$\mathbf{2.83 \times 10^6}$
3	$\mathbf{0.9583 \pm 0.006}$	$\mathbf{2.0 \times 10^6}$	0.9553 ± 0.001	6.8×10^6
4	0.9582 ± 0.008	$\mathbf{1.0 \times 10^6}$	$\mathbf{0.9592 \pm 0.002}$	22.7×10^6

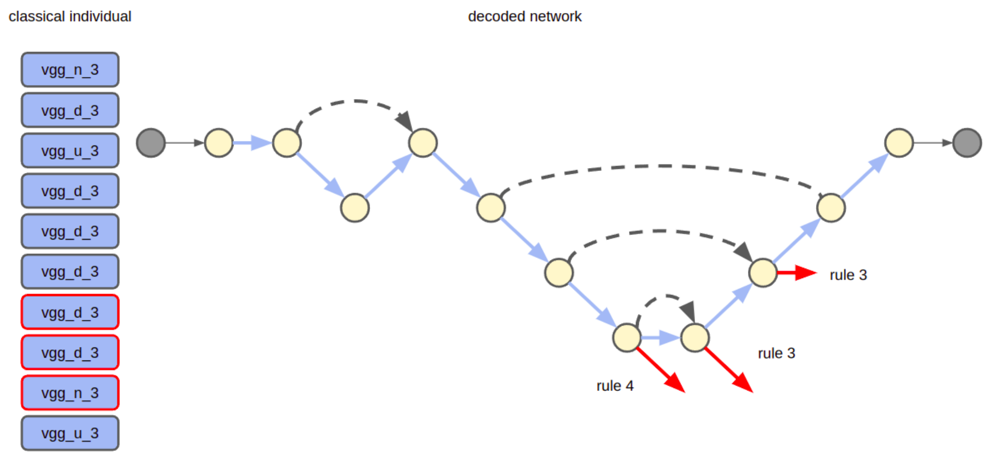

Figure 6.1: Classical individual and decoded network for experiment 1 on the spleen dataset.

cell level ($D = 4$), indicating that the features from the spleen dataset are more localized and therefore do not need that amount of reduction operations. The early upsampling also indicates that features from less downscaled feature maps contain valuable information for spleen segmentation. In the Figure, we can also notice that rules 3 and 4 were applied during the decoding process. The modified functions are highlighted in red.

Figure 6.2 shows the architecture found in experiment 2. On the block level, we notice a preference for ResNet blocks. The same preference is also seen in the network found in experiment 3 (Figure 6.3).

The architecture found in experiment 4 (Figure 6.4) is the most different among the experiments. It achieves only cell level $d = 2$, corroborating the idea that for the spleen segmentation, the features are more local and do not require many reduction operations. Again we verify the same preference for ResNet blocks together with DenseNet blocks. Those blocks are characterized by residual connections.

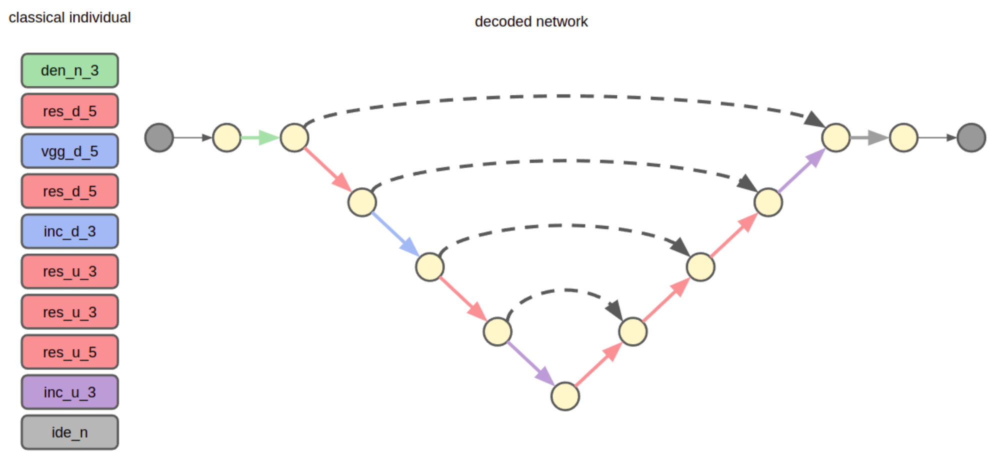

Figure 6.2: Classical individual and decoded network for experiment 2 on the spleen dataset.

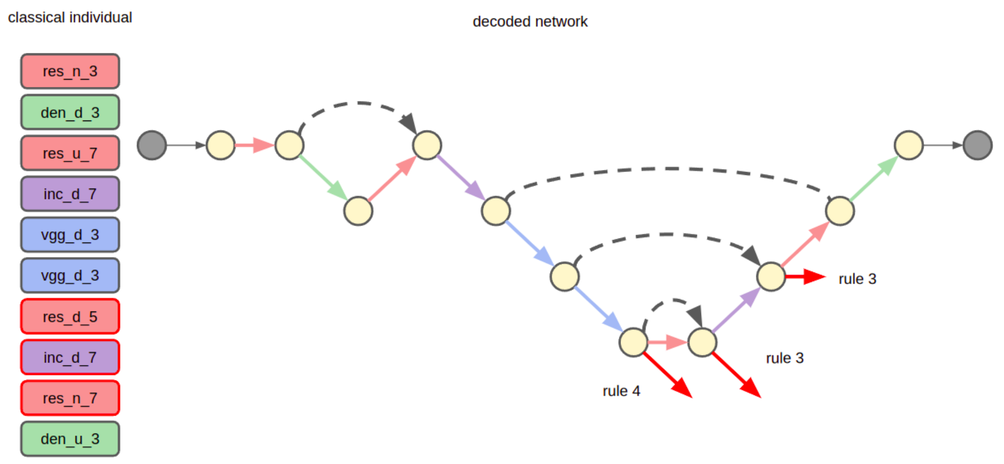

Figure 6.3: Classical individual and decoded network for experiment 3 on the spleen dataset.

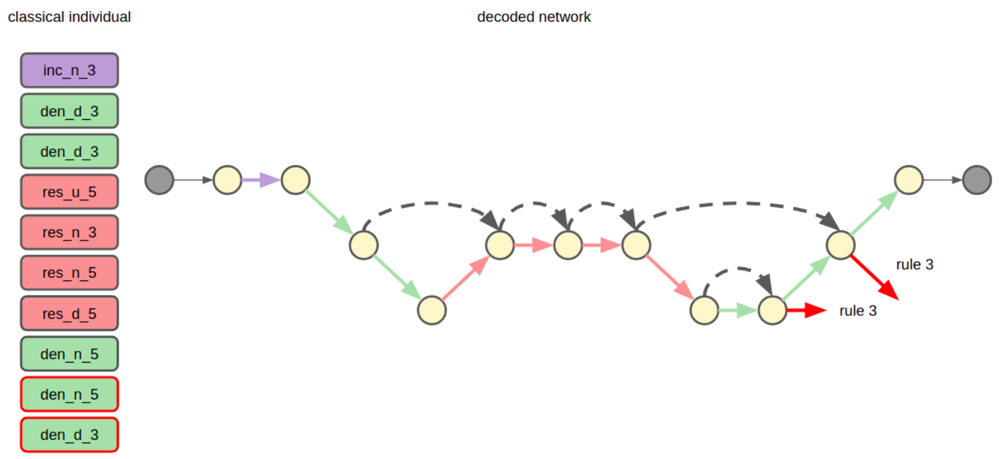

Figure 6.4: Classical individual and decoded network for experiment 4 on the spleen dataset.

6.4.2 Prostate Dataset

The results of the experiments on the prostate dataset are summarized in Table 6.4.

Table 6.4: Prostate Dataset Experiments Results

Experiment	**DSC**	**#Params**	**GPU days**
1	0.6598 ± 0.070	3.8×10^6	**1.61**
2	0.6682 ± 0.066	5.7×10^6	1.74
3	$\mathbf{0.6887 \pm 0.067}$	17.0×10^6	1.68
4	0.6821 ± 0.067	16.4×10^6	1.83
U-Net	0.6529 ± 0.070	$\mathbf{2.7 \times 10^6}$	NA
ResU-Net	0.6467 ± 0.069	2.8×10^6	NA

We see also that SegQNAS was able to find architectures in all experiments that achieved better DSC than the baselines. However, this time, all architectures found have more parameters than the baselines. The best architecture was also found by experiment 3 and achieved a very similar performance compared to experiment 4.

Table 6.5 contains the comparison for each experiment with MixedBlocks results.

Table 6.5: Prostate Dataset Experiments Results comparison with MixedBlock [85]

	SegQNAS		**MixedBlock** [85]	
Experiment	**DSC**	**#Params**	**DSC**	**#Params**
1	$\mathbf{0.6598 \pm 0.070}$	3.8×10^6	0.6593 ± 0.004	$\mathbf{3.4 \times 10^6}$
2	0.6682 ± 0.066	$\mathbf{5.7 \times 10^6}$	$\mathbf{0.6796 \pm 0.007}$	22.7×10^6
3	$\mathbf{0.6887 \pm 0.067}$	17.0×10^6	0.6702 ± 0.005	$\mathbf{6.7 \times 10^6}$
4	$\mathbf{0.6821 \pm 0.067}$	16.4×10^6	0.6760 ± 0.010	$\mathbf{3.0 \times 10^6}$

It is possible to notice that SegQNAS found better-performing architecture for experiments 1, 3, and 4 in terms of DSCs. However, it found a network with much more parameters. It is interesting to notice that again for experiment 2, MixedBlock outperformed SegQNAS.

Figure 6.5 shows the architecture found in experiment 1. The topology is very similar to the U-Net one, however, the nonscaling operations happen in the second and sixth nodes instead of the first and last. We also notice that rules 2 and 3 were applied in the last 5 nodes. It shows that the search

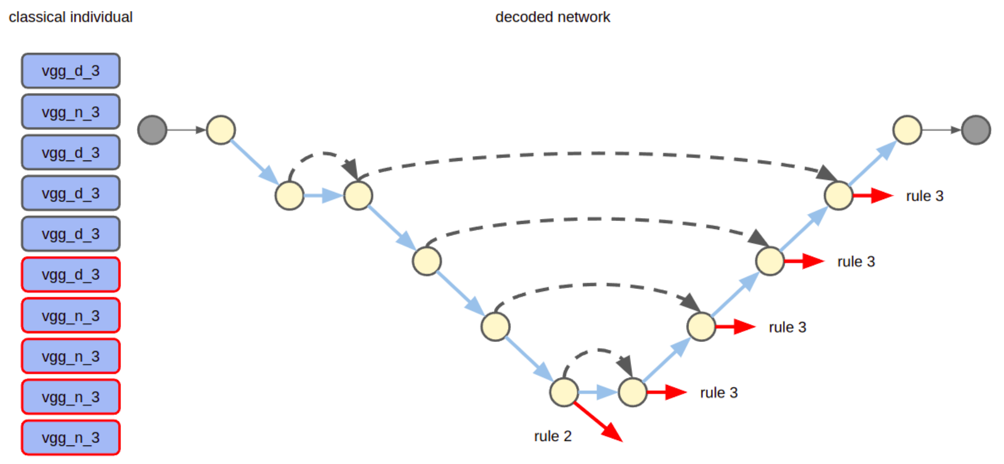

Figure 6.5: Classical individual and decoded network for experiment 1 on the prostate dataset.

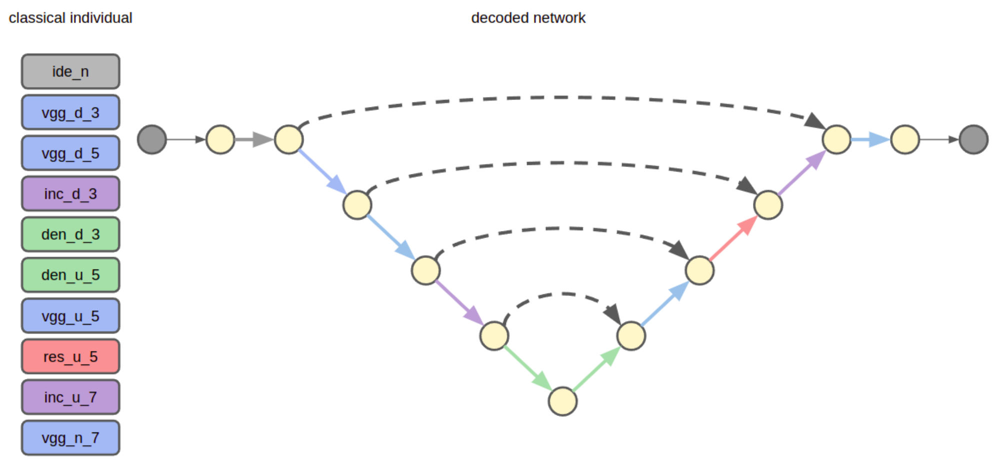

Figure 6.6: Classical individual and decoded network for experiment 2 on the prostate dataset.

favored more operations on the higher cell levels instead of performing the upscaling. This can indicate that features from higher cell levels (that contain more global information) are important for prostate segmentation in contrast to spleen segmentation.

That conclusion is also corroborated by the architecture found in experiment 2 (Figure 6.6). In that case, we notice that dense cells, which perform more convolutional operations, are selected for the cells on the highest cell level.

In the architecture found by experiment 3 (Figure 6.7), we notice that the search process favored kernels with greater size and, therefore, convolutions with greater reception fields. That is also indicative that more global information is important for prostate segmentation.

For experiment 4 (Figure 6.8), the architecture found was very similar

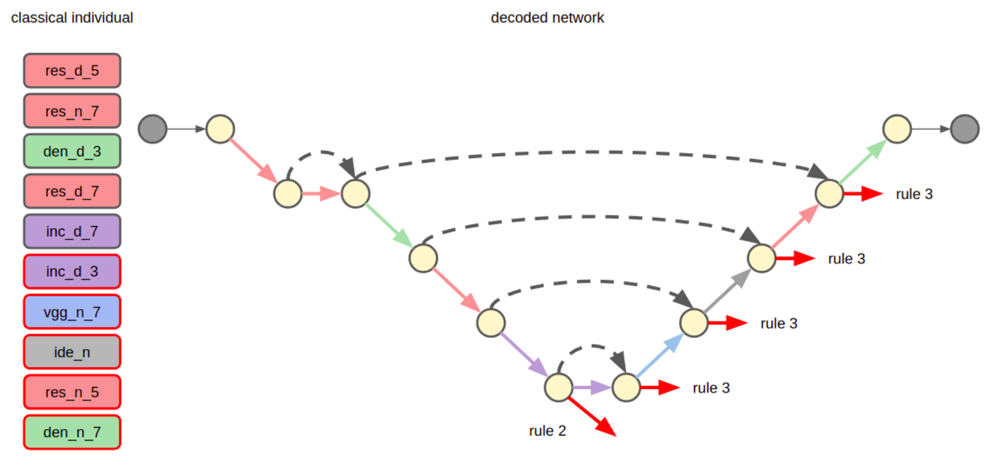

Figure 6.7: Classical individual and decoded network for experiment 3 on the prostate dataset.

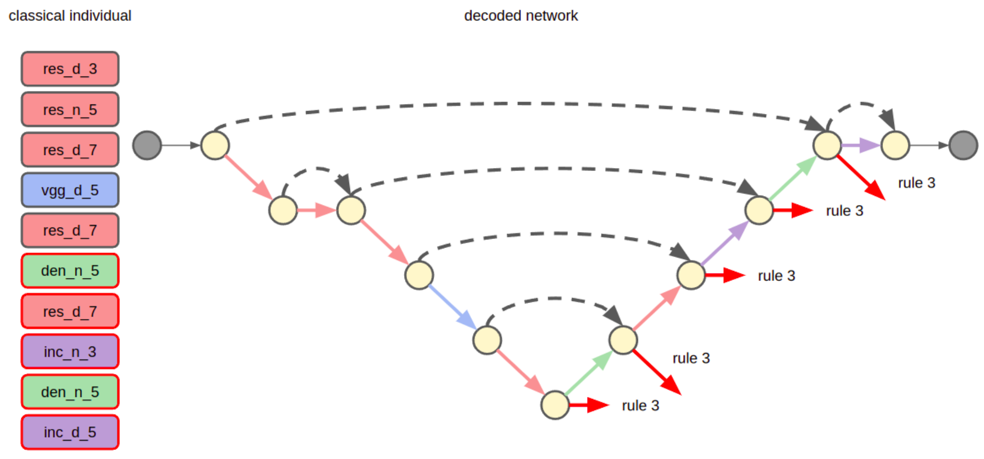

Figure 6.8: Classical individual and decoded network for experiment 4 on the prostate dataset.

to the one found in experiment 1 and used in experiment 3. And it is also similar to the U-Net architecture. It is possible to notice the same behavior of having decoding rules applied to the last 5 nodes. Another similarity between the network from experiments 3 and 4 is that they have only one VGG block and presented a preference for more complex blocks, especially ResNet blocks.

One important thing we must notice is that for experiments 1 and 4, in which the topology search was performed, no upscaling cell was selected.

7
Conclusions

The primary goal of this work was to propose a NAS algorithm, based on Q-NAS, that is able to search for semantic segmentation DNN architectures. The proposed solution, SegQNAS, uses the same QIEA as Q-NAS but with a different search space, decoding process, and evaluation strategy. SegQNAS was able to perform NAS for semantic segmentation architectures on a provided dataset.

The search space used is a global search space comprising a two-level representation, the topology level and the cell levels. This representation allows for great flexibility in defining parameters for the topology and blocks for the cells. In our work, we limited the blocks to a set of four well-known blocks from the literature, however, more possibilities could be explored.

Regarding the topology level, some design constraint rules were enforced during the decoding process to ensure the candidate architectures were feasible.

The secondary goal was to evaluate the architectures found by SegQNAS on two benchmark datasets. To do so, in the experiments, we explored the search space in different ways. First, experiment 1 covers only the topology search space, keeping the blocks fixed to a default. Then, experiment 2 explores performing the search on the cell search space while keeping the topology fixed. Experiments 3 and 4 performed the search on the whole search space (both topology and cell level search space). Experiment 3 approached the search as a bi-level search, where the topology search is performed first and then the cell search is performed considering the previously found topology. Finally, experiment 4 explored the search on the topology and cell level search spaces simultaneously.

The results from the experiments showed that SegQNAS was able to discover architectures that outperformed both hand-crafted baseline architectures, namely the U-Net and ResU-Net. The models for semantic segmentation evolved by the SegQNAS methodology also showed competitive results to the NAS literature, both in terms of DSC and the number of parameters of the networks, when compared to similar work.

The topologies and block choices in the experiments revealed the task-specificity of the search. Architectures with different characteristics were found for the spleen and prostate datasets. That indicates that the extraction of features for the two datasets required different architectures in terms of topologies and cells.

It is important to point out that both the training of the candidate architectures and the evolution process of the QIEA are stochastic processes. To mitigate the non-deterministic nature of the training, a 3 initialization 5-fold cross-validation took place to evaluate each candidate architecture. In order to mitigate this non-deterministic effect on the evolution itself we would have to perform the experiment multiple times and get the average of the results. That would lead to a more reliable interpretation of the achieved results. However, that was not possible due to resource constraints.

For future works, several directions can be explored. Firstly, a comparison with a random search baseline could be conducted to evaluate the efficiency and effectiveness of SegQNAS. Additionally, incorporating multi-objective optimization techniques would allow for exploring trade-offs between accuracy, model complexity, and runtime efficiency. The current rules in the decoding process are deterministic and could restrict the search space exploration. Therefore, introducing stochastic elements into the decoding rules process could enhance exploration capabilities and lead to novel architectural configurations.

Furthermore, future research could include a qualitative analysis of the discovered architectures, visualizing and interpreting learned representations and feature maps, and incorporating explainable AI techniques. This could provide insights into the decision-making process of the model and refine the search space and decoding process.

Expanding the search space to include more blocks from the literature would enable the exploration of a wider range of architectural configurations. Finally, investigating different parameter configurations for the search space would provide insights into the sensitivity of SegQNAS to architectural choices and guide the refinement of the search space design.

Bibliography

[1] ELSKEN, T.; METZEN, J. H. ; HUTTER, F.. **Neural architecture search: A survey**. The Journal of Machine Learning Research, 20(1):1997–2017, 2019.

[2] LIU, Y.; SUN, Y.; XUE, B.; ZHANG, M.; YEN, G. G. ; TAN, K. C.. **A survey on evolutionary neural architecture search**. IEEE transactions on neural networks and learning systems, 2021.

[3] SZWARCMAN, D.; CIVITARESE, D. ; VELLASCO, M.. **Quantum-inspired evolutionary algorithm applied to neural architecture search**. Applied Soft Computing, 120:108674, 2022.

[4] ANTONELLI, M.; REINKE, A.; BAKAS, S.; FARAHANI, K.; KOPP-SCHNEIDER, A.; LANDMAN, B. A.; LITJENS, G.; MENZE, B.; RONNEBERGER, O.; SUMMERS, R. M. ; OTHERS. **The medical segmentation decathlon**. Nature communications, 13(1):1–13, 2022.

[5] MINAEE, S.; BOYKOV, Y. Y.; PORIKLI, F.; PLAZA, A. J.; KEHTARNAVAZ, N. ; TERZOPOULOS, D.. **Image segmentation using deep learning: A survey**. IEEE transactions on pattern analysis and machine intelligence, 2021.

[6] KIRILLOV, A.; HE, K.; GIRSHICK, R.; ROTHER, C. ; DOLLAR, P.. **Panoptic segmentation**. In: PROCEEDINGS OF THE IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), June 2019.

[7] ZHAO, H.; SHI, J.; QI, X.; WANG, X. ; JIA, J.. **Pyramid scene parsing network**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), July 2017.

[8] FENG, D.; HAASE-SCHÜTZ, C.; ROSENBAUM, L.; HERTLEIN, H.; GLAESER, C.; TIMM, F.; WIESBECK, W. ; DIETMAYER, K.. **Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges**. IEEE Transactions on Intelligent Transportation Systems, 22(3):1341–1360, 2020.

[9] MA, L.; LIU, Y.; ZHANG, X.; YE, Y.; YIN, G. ; JOHNSON, B. A.. **Deep learning in remote sensing applications: A meta-analysis and review**. ISPRS journal of photogrammetry and remote sensing, 152:166–177, 2019.

[10] ABDULLAH, F.; JALAL, A.. **Semantic segmentation based crowd tracking and anomaly detection via neuro-fuzzy classifier in smart surveillance system**. Arabian Journal for Science and Engineering, p. 1–18, 2022.

[11] SUGANYADEVI, S.; SEETHALAKSHMI, V. ; BALASAMY, K.. **A review on deep learning in medical image analysis**. International Journal of Multimedia Information Retrieval, 11(1):19–38, 2022.

[12] OTSU, N.. **A threshold selection method from gray-level histograms**. IEEE transactions on systems, man, and cybernetics, 9(1):62–66, 1979.

[13] NOCK, R.; NIELSEN, F.. **Statistical region merging**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(11):1452–1458, 2004.

[14] DHANACHANDRA, N.; MANGLEM, K. ; CHANU, Y. J.. **Image segmentation using k-means clustering algorithm and subtractive clustering algorithm**. Procedia Computer Science, 54:764–771, 2015.

[15] NAJMAN, L.; SCHMITT, M.. **Watershed of a continuous function**. Signal Processing, 38(1):99–112, 1994. Mathematical Morphology and its Applications to Signal Processing.

[16] MICHAEL KASS, A. W.; TERZOPOULOS, D.. **Snakes: Active contour models**. International Journal of Computer Vision, 1:321–331, 1998.

[17] BOYKOV, Y.; VEKSLER, O. ; ZABIH, R.. **Fast approximate energy minimization via graph cuts**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(11):1222–1239, 2001.

[18] PLATH, N.; TOUSSAINT, M. ; NAKAJIMA, S.. **Multi-class image segmentation using conditional random fields and global classification**. In: PROCEEDINGS OF THE 26TH ANNUAL INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 817–824, 2009.

[19] HOCHBAUM, D. S.. **An efficient algorithm for image segmentation, markov random fields and related problems**. Journal of the ACM (JACM), 48(4):686–701, 2001.

[20] STARCK, J.-L.; ELAD, M. ; DONOHO, D. L.. **Image decomposition via the combination of sparse representations and a variational approach**. IEEE transactions on image processing, 14(10):1570–1582, 2005.

[21] MINAEE, S.; WANG, Y.. **An admm approach to masked signal decomposition using subspace representation**. IEEE Transactions on Image Processing, 28(7):3192–3204, 2019.

[22] GHOSH, S.; DAS, N.; DAS, I. ; MAULIK, U.. **Understanding deep learning techniques for image segmentation**. ACM Computing Surveys (CSUR), 52(4):1–35, 2019.

[23] FUKUSHIMA, K.; MIYAKE, S. ; ITO, T.. **Neocognitron: A neural network model for a mechanism of visual pattern recognition**. IEEE transactions on systems, man, and cybernetics, (5):826–834, 1983.

[24] LECUN, Y.; BOSER, B.; DENKER, J.; HENDERSON, D.; HOWARD, R.; HUBBARD, W. ; JACKEL, L.. **Handwritten digit recognition with a back-propagation network**. In: Touretzky, D., editor, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, volumen 2. Morgan-Kaufmann, 1989.

[25] LECUN, Y.; BOTTOU, L.; BENGIO, Y. ; HAFFNER, P.. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[26] LEARNING, D.; GOODFELLOW, I.; BENGIO, Y. ; COURVILLE, A.. **Adaptive computation and machine learning series**, 2016.

[27] LECUN, Y.; BENGIO, Y.; HINTON, G. ; OTHERS. **Deep learning. nature, 521 (7553), 436-444**. Google Scholar Google Scholar Cross Ref Cross Ref, p. 25, 2015.

[28] KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G. E.. **Imagenet classification with deep convolutional neural networks**. Communications of the ACM, 60(6):84–90, 2017.

[29] SIMONYAN, K.; ZISSERMAN, A.. **Very deep convolutional networks for large-scale image recognition**. arXiv preprint arXiv:1409.1556, 2014.

[30] SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J. ; WOJNA, Z.. **Rethinking the inception architecture for computer vision**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 2818–2826, 2016.

[31] HE, K.; ZHANG, X.; REN, S. ; SUN, J.. **Deep residual learning for image recognition**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 770–778, 2016.

[32] HUANG, G.; LIU, Z.; VAN DER MAATEN, L. ; WEINBERGER, K. Q.. **Densely connected convolutional networks**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 4700–4708, 2017.

[33] LONG, J.; SHELHAMER, E. ; DARRELL, T.. **Fully convolutional networks for semantic segmentation**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 3431–3440, 2015.

[34] NOH, H.; HONG, S. ; HAN, B.. **Learning deconvolution network for semantic segmentation**. In: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV), December 2015.

[35] BADRINARAYANAN, V.; KENDALL, A. ; CIPOLLA, R.. **Segnet: A deep convolutional encoder-decoder architecture for image segmentation**. IEEE transactions on pattern analysis and machine intelligence, 39(12):2481–2495, 2017.

[36] RONNEBERGER, O.; FISCHER, P. ; BROX, T.. **U-net: Convolutional networks for biomedical image segmentation**. In: MEDICAL IMAGE COMPUTING AND COMPUTER-ASSISTED INTERVENTION–MICCAI 2015: 18TH INTERNATIONAL CONFERENCE, MUNICH, GERMANY, OCTOBER 5-9, 2015, PROCEEDINGS, PART III 18, p. 234–241. Springer, 2015.

[37] ZHOU, Z.; RAHMAN SIDDIQUEE, M. M.; TAJBAKHSH, N. ; LIANG, J.. **Unet++: A nested u-net architecture for medical image segmentation**. In: DEEP LEARNING IN MEDICAL IMAGE ANALYSIS AND MULTIMODAL LEARNING FOR CLINICAL DECISION SUPPORT: 4TH INTERNATIONAL WORKSHOP, DLMIA 2018, AND 8TH INTERNATIONAL WORKSHOP, ML-CDS 2018, HELD IN CONJUNCTION WITH MICCAI 2018, GRANADA, SPAIN, SEPTEMBER 20, 2018, PROCEEDINGS 4, p. 3–11. Springer, 2018.

[38] MILLETARI, F.; NAVAB, N. ; AHMADI, S.-A.. **V-net: Fully convolutional neural networks for volumetric medical image segmentation**. In: 2016 FOURTH INTERNATIONAL CONFERENCE ON 3D VISION (3DV), p. 565–571. Ieee, 2016.

[39] JÉGOU, S.; DROZDZAL, M.; VAZQUEZ, D.; ROMERO, A. ; BENGIO, Y.. **The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION WORKSHOPS, p. 11–19, 2017.

[40] WISTUBA, M.; RAWAT, A. ; PEDAPATI, T.. **A survey on neural architecture search**. arXiv preprint arXiv:1905.01392, 2019.

[41] BAKER, B.; GUPTA, O.; NAIK, N. ; RASKAR, R.. **Designing neural network architectures using reinforcement learning**. arXiv preprint arXiv:1611.02167, 2016.

[42] BARRET, Z.; LE QUOC, V. ; OTHERS. **Neural architecture search with reinforcement learning**. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATOINS, volumen 1, 2017.

[43] SUGANUMA, M.; SHIRAKAWA, S. ; NAGAO, T.. **A genetic programming approach to designing convolutional neural network architectures**. In: PROCEEDINGS OF THE GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, p. 497–504, 2017.

[44] ELSKEN, T.; METZEN, J. H. ; HUTTER, F.. **Efficient multi-objective neural architecture search via lamarckian evolution**. arXiv preprint arXiv:1804.09081, 2018.

[45] BROCK, A.; LIM, T.; RITCHIE, J. M. ; WESTON, N.. **Smash: one-shot model architecture search through hypernetworks**. arXiv preprint arXiv:1708.05344, 2017.

[46] ZOPH, B.; VASUDEVAN, V.; SHLENS, J. ; LE, Q. V.. **Learning transferable architectures for scalable image recognition**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 8697–8710, 2018.

[47] ZHONG, Z.; YAN, J.; WU, W.; SHAO, J. ; LIU, C.-L.. **Practical blockwise neural network architecture generation**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 2423–2432, 2018.

[48] LIU, H.; SIMONYAN, K.; VINYALS, O.; FERNANDO, C. ; KAVUKCUOGLU, K.. **Hierarchical representations for efficient architecture search**. arXiv preprint arXiv:1711.00436, 2017.

[49] WILLIAMS, R. J.. **Simple statistical gradient-following algorithms for connectionist reinforcement learning.** Reinforcement learning, p. 5–32, 1992.

[50] CAI, H.; CHEN, T.; ZHANG, W.; YU, Y. ; WANG, J.. **Efficient architecture search by network transformation.** In: PROCEEDINGS OF THE AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, volumen 32, 2018.

[51] DE JONG, K.. **Evolutionary computation: a unified approach.** In: PROCEEDINGS OF THE 2016 ON GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE COMPANION, p. 185–199, 2016.

[52] REAL, E.; MOORE, S.; SELLE, A.; SAXENA, S.; SUEMATSU, Y. L.; TAN, J.; LE, Q. V. ; KURAKIN, A.. **Large-scale evolution of image classifiers.** In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 2902–2911. PMLR, 2017.

[53] BEYER, H.-G.; SCHWEFEL, H.-P.. **Evolution strategies–a comprehensive introduction.** Natural computing, 1:3–52, 2002.

[54] REAL, E.; AGGARWAL, A.; HUANG, Y. ; LE, Q. V.. **Regularized evolution for image classifier architecture search.** In: PROCEEDINGS OF THE AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, volumen 33, p. 4780–4789, 2019.

[55] JIN, H.; SONG, Q. ; HU, X.. **Auto-keras: An efficient neural architecture search system.** In: PROCEEDINGS OF THE 25TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY & DATA MINING, p. 1946–1956, 2019.

[56] LIU, C.; CHEN, L.-C.; SCHROFF, F.; ADAM, H.; HUA, W.; YUILLE, A. L. ; FEI-FEI, L.. **Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation.** In: PROCEEDINGS OF THE IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 82–92, 2019.

[57] WISTUBA, M.. **Finding competitive network architectures within a day using uct.** arXiv preprint arXiv:1712.07420, 2017.

[58] ELSKEN, T.; METZEN, J.-H. ; HUTTER, F.. **Simple and efficient architecture search for convolutional neural networks.** arXiv preprint arXiv:1711.04528, 2017.

[59] LIU, H.; SIMONYAN, K. ; YANG, Y.. **Darts: Differentiable architecture search**. arXiv preprint arXiv:1806.09055, 2018.

[60] XIE, S.; ZHENG, H.; LIU, C. ; LIN, L.. **Snas: stochastic neural architecture search**. arXiv preprint arXiv:1812.09926, 2018.

[61] CAI, H.; ZHU, L. ; HAN, S.. **Proxylessnas: Direct neural architecture search on target task and hardware**. arXiv preprint arXiv:1812.00332, 2018.

[62] SHIN, R.; PACKER, C. ; SONG, D.. **Differentiable neural network architecture search**. 2018.

[63] AHMED, K.; TORRESANI, L.. **Maskconnect: Connectivity learning by gradient descent**. In: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV), p. 349–365, 2018.

[64] ZHOU, Y.; EBRAHIMI, S.; ARIK, S. Ö.; YU, H.; LIU, H. ; DIAMOS, G.. **Resource-efficient neural architect**. arXiv preprint arXiv:1806.07912, 2018.

[65] ZELA, A.; KLEIN, A.; FALKNER, S. ; HUTTER, F.. **Towards automated deep learning: Efficient joint neural architecture and hyperparameter search**. arXiv preprint arXiv:1807.06906, 2018.

[66] KLEIN, A.; FALKNER, S.; BARTELS, S.; HENNIG, P. ; HUTTER, F.. **Fast bayesian optimization of machine learning hyperparameters on large datasets**. In: ARTIFICIAL INTELLIGENCE AND STATISTICS, p. 528–536. PMLR, 2017.

[67] CHRABASZCZ, P.; LOSHCHILOV, I. ; HUTTER, F.. **A downsampled variant of imagenet as an alternative to the cifar datasets**. arXiv preprint arXiv:1707.08819, 2017.

[68] RAWAL, A.; MIIKKULAINEN, R.. **From nodes to networks: Evolving recurrent neural networks**. arXiv preprint arXiv:1803.04439, 2018.

[69] DOMHAN, T.; SPRINGENBERG, J. T. ; HUTTER, F.. **Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves**. In: TWENTY-FOURTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2015.

[70] SWERSKY, K.; SNOEK, J. ; ADAMS, R. P.. **Freeze-thaw bayesian optimization**. arXiv preprint arXiv:1406.3896, 2014.

[71] KLEIN, A.; FALKNER, S.; SPRINGENBERG, J. T. ; HUTTER, F.. **Learning curve prediction with bayesian neural networks**. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 2017.

[72] BAKER, B.; GUPTA, O.; RASKAR, R. ; NAIK, N.. **Accelerating neural architecture search using performance prediction**. arXiv preprint arXiv:1705.10823, 2017.

[73] LIU, C.; ZOPH, B.; NEUMANN, M.; SHLENS, J.; HUA, W.; LI, L.-J.; FEI-FEI, L.; YUILLE, A.; HUANG, J. ; MURPHY, K.. **Progressive neural architecture search**. In: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV), p. 19–34, 2018.

[74] WEI, T.; WANG, C.; RUI, Y. ; CHEN, C. W.. **Network morphism**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 564–572. PMLR, 2016.

[75] JIN, H.; SONG, Q. ; HU, X.. **Auto-keras: Efficient neural architecture search with network morphism**. arXiv preprint arXiv:1806.10282, 5, 2018.

[76] DUSHATSKIY, A.; ALDERLIESTEN, T. ; BOSMAN, P. A.. **Heed the noise in performance evaluations in neural architecture search**. In: PROCEEDINGS OF THE GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE COMPANION, p. 2104–2112, 2022.

[77] CHEN, L.-C.; COLLINS, M.; ZHU, Y.; PAPANDREOU, G.; ZOPH, B.; SCHROFF, F.; ADAM, H. ; SHLENS, J.. **Searching for efficient multi-scale architectures for dense image prediction**. Advances in neural information processing systems, 31, 2018.

[78] CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K. ; YUILLE, A. L.. **Semantic image segmentation with deep convolutional nets and fully connected crfs**. arXiv preprint arXiv:1412.7062, 2014.

[79] CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K. ; YUILLE, A. L.. **Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs**. IEEE transactions on pattern analysis and machine intelligence, 40(4):834–848, 2017.

[80] CHEN, L.-C.; PAPANDREOU, G.; SCHROFF, F. ; ADAM, H.. **Rethinking atrous convolution for semantic image segmentation**. arXiv preprint arXiv:1706.05587, 2017.

[81] SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A. ; CHEN, L.-C.. **Mobilenetv2: Inverted residuals and linear bottlenecks**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 4510–4520, 2018.

[82] CHOLLET, F.. **Xception: Deep learning with depthwise separable convolutions**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 1251–1258, 2017.

[83] GOLOVIN, D.; SOLNIK, B.; MOITRA, S.; KOCHANSKI, G.; KARRO, J. ; SCULLEY, D.. **Google vizier: A service for black-box optimization**. In: PROCEEDINGS OF THE 23RD ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 1487–1495, 2017.

[84] YU, Q.; YANG, D.; ROTH, H.; BAI, Y.; ZHANG, Y.; YUILLE, A. L. ; XU, D.. **C2fnas: Coarse-to-fine neural architecture search for 3d medical image segmentation**. In: PROCEEDINGS OF THE IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 4126–4135, 2020.

[85] BOSMA, M. M.; DUSHATSKIY, A.; GREWAL, M.; ALDERLIESTEN, T. ; BOSMAN, P.. **Mixed-block neural architecture search for medical image segmentation**. In: MEDICAL IMAGING 2022: IMAGE PROCESSING, volumen 12032, p. 193–199. SPIE, 2022.

[86] WENG, Y.; ZHOU, T.; LI, Y. ; QIU, X.. **Nas-unet: Neural architecture search for medical image segmentation**. IEEE access, 7:44247–44257, 2019.

[87] CHOLLET, F.; OTHERS. **Keras**. `https://keras.io`, 2015.