A Symmetric grammar approach for designing segmentation models

1st Ricardo H. R. Lima, 2nd Aurora Pozo
Computer Science Department
Federal University of Paraná
Curitiba, Brazil
{rhrlima, aurora}@inf.ufpr.br

3rd Alexander Mendiburu, 4th Roberto Santana
Dept. of Computer Science and Artificial Intelligence
University of the Basque Country UPV/EHU
Donostia/San Sebastian, Spain
{alexander.mendiburu, roberto.santana}@ehu.eus

***Abstract*—Image segmentation is a relevant problem in computer vision present in multiple application domains. One of the most used methods for image segmentation is U-net, a type of convolutional network with additional constraints in its architecture. Studies regarding the U-net usually rely on well-known architectures, which leads to a narrow exploration of the possibilities, and possibly impacting the performance. Genetic Programming approaches have become increasingly popular for designing neural networks due to studies where the generated models were able to achieve results comparable to humans. These approaches can evolve the structure at different levels of abstraction, reducing the need for a specialist. In this paper, we propose the use of Grammatical Evolution for evolving U-net architectures. We propose a mirror grammar, which is capable of generating a variety of flexible U-nets that better explores the search space. We show that the proposed grammar can capture the complex constraints that define the U-nets and achieve comparable results in terms of accuracy, on a benchmark of segmentation problems of varying difficulty.**

***Index Terms*—grammatical evolution, automatic design, deep neural networks, u-net**

I. Introduction

Neuroevolutionary design of deep neural networks is an increasingly used method in Machine Learning (ML) applications. Recently, many studies have been developed for the automatic design of Deep Neural Networks (DNNs) [2], [18], [37]. A common approach is to express the architecture of a network considering its layers as building blocks. Combining different types of building blocks allows the generation of an infinite amount of novel designs, where the objective is to improve the overall performance of the network. Measuring this performance is a difficult task. Most studies focus on applying small changes to a well-known architecture so it can achieve high accuracy on the target problem. This path leads to a poor exploration of the space of architectures, ignoring possibly better designs that might differ from the established state-of-the-art networks, for instance, Convolutional Neural Networks (CNNs) [15] for images, Long-short Term Memory Networks (LSTM) [5] for texts, and Recurrent Neural Networks (RNN) [8] for audio problems.

Segmenting an image consists of dividing it in multiple segments of pixels to describe objects. The objective is to simplify the information, modifying its representation into something more useful, and easier to analyze. The U-net [32] is the most popular deep learning model used for image segmentation. It is composed basically of convolutional and pooling layers. The non-sequential organization of its layers is one of the most relevant characteristics, which gives the network the ability to re-use information from previous layers in order to learn high-resolution features. These characteristics add new restrictions, which make the design of U-nets harder, compared to classical CNNs.

Usually, each network is manually configured for each specific problem by a specialist, who will select the building blocks for the architecture. Most current studies on U-net applications [16], [26], [34] rely on using and modifying well-known architectures, leading them to explore only a small portion of the search space. The use of techniques such as Genetic Programming (GP) can offer an easy and flexible approach to design a wider variety of architectures.

The most popular approaches to the automatic design rely on Genetic Programming (GP) techniques and its variations [14], [28], [33]. GP is an extension to the Genetic Algorithm (GA) [30], where the candidate solutions are executable programs rather than fixed-length strings. The search space is described from the set of components defined by the user, to compose the programs. Moreover, the search engine operates by creating a population of randomly generated solutions that are selected, recombined, and evaluated in order to select the best solution found throughout a given number of generations.

In this work, we propose the use of Dynamic Structured Grammatical Evolution (DSGE) [2] to design neural networks for image segmentation tasks. DSGE is a version of GP that uses grammars to define the space of programs, i.e., the set of components. The grammar is a set of symbols and rules that are used to guide the search, being very flexible and easy to adapt to different domains. Thus, we propose to study the use of GE to design U-nets, in order to verify if it is possible to find solutions that balance novelty and performance while proposing a more efficient exploration of the space. GE has also been recently applied to the evolution of deep neural networks that follow a sequential architecture [18], different from the U-net that follows a non-sequential design. The proposed experiments will evaluate the approach on a segmentation task, based on generated datasets with increasing levels of difficulty. In addition, as one of the main drawbacks

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

of these proposals is the time required to evaluate each solution (train a U-net), we have tested two approaches: 1) evolve the networks while training them during the evolution; and 2) evolve the networks generated with random weights, training only the final best.

This work is organized as follows. Section II covers previous neuroevolutionary approaches and U-net background. Section III presents some foundations for the DSGE technique and how it is used to design neural networks. Our approach is presented in Section IV, with more details about the grammar we propose to design U-nets. Next, we describe the performed experiments in Section V, including a description of the way in which our approach is evaluated and its contributions to the design of U-nets. Finally, in Section VI, we cover a summary of the work discussing the results and proposing future directions to this study.

II. Related Work

A. NeuroEvolution

Automatically design and configure programs emerged as an essential research direction [11], [12], [20]. Evolutionary approaches have been widely used to automatically design algorithms aiming to find the best combination of parameters as well as proposing novel architectures [4], [17], [25], [29].

Neuroevolutionary approaches to evolve CNNs are extensive; we only cover some of the most closely related work. We consider the following criteria to review the algorithms and highlight the differences and similarities with our proposal:

1) The evolutionary approach applied to evolve the network;
2) Evolution of CNNs;
3) Structural or topological constraints of the evolved architectures.

Initial studies regarding the evolution of ANNs covered the evolution of only weights [23], [41], and further followed by evolving the topology [7], [31]. Moreover, some neuroevolutionary approaches simultaneously evolve topologies and weights [13], [36], [39]. The approach we propose in this paper evolves the topologies only, while the weights are learned from data or generated using a different technique.

Regarding the design of CNNs, the most common approaches are Evolutionary Algorithms (EAs), for instance, GP approaches [14], [24], [33], GAs [41], and other evolutionary strategies [6]. Only recently, works on GE have been proposed for CNNs [2], [17], [19]. Our GE approach follows a similar path, however, we extend the grammar by proposing a novel way to generate the networks, in order to cover a more extensive portion of the design space, while trying to minimize invalid solutions.

In terms of the CNN topological characteristics, almost all the previous methods have focused on sequential architectures [10], [17] or variants of well known CNN architectures. Novel approaches have been proposed, such as Residual Networks [10] with the “skip-connections”, or more robust modules, as in Google Inception network [38].

In this study, we show how to use DSGE encoding to design a network effectively. We are also proposing a flexible grammar structure that allows managing the constraints of the network, which later could be expanded by the addition of novelty blocks, for example, skip-connections and inception modules. To the best knowledge of the authors, this type of approach has not been previously reported in the literature.

B. U-nets

U-nets [32] were proposed as a way to overcome the limitations of CNNs when dealing with the segmentation of images. While mainly applied to image segmentation, U-nets are also an essential component to generative deep learning approaches [9], [27]. Classical CNNs have their success limited due to the size of both the network and the available training sets. In this sense, U-nets can take advantage of smaller datasets to achieve higher accuracy values.

The U-net architecture (Figure 1) is mainly composed of convolutional and max-pooling layers, and it is divided into three parts: the contracting path (left side), the expansive path (right side), and the middle part that connects the two previous ones. The contracting path follows a typical convolutional network scheme, with repeated applications of two 3x3 convolutions, each followed by a ReLU activation and a 2x2 max pooling with stride 2 for down-sampling. At each down-sampling step, the number of filters is doubled. On the expansive path, each step consists of an up-sampling followed by a 2x2 convolution that halves the number of filters, a concatenation with the corresponding cropped image from the contracting path, and two 3x3 convolutions followed by a ReLU activation. The middle part is composed mostly of convolutional layers and a few dropout layers. A 1x1 convolutional layer performs classification, which maps each of the pixels into one of the defined classes.

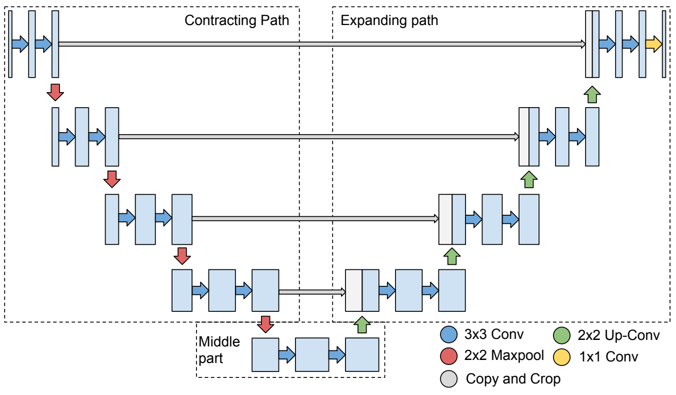

Fig. 1. Vanilla U-Net architecture.

By analyzing the U-net architecture, it is possible to list some characteristics:

- For every down-sample step, derived from a pooling layer on the contracting path, there is an up-sample step on the expanding path;

- The last layer before a down-sample step is connected to the first layer after the up-sample step by a concatenation layer, where the image size matches;
- At each down or up-sample, the parameters decrease or increase within the same factor.

In general, most U-net applications are usually tied to the original design [16], [26], [40]. Besides, works that proposed modifications have presented exciting outcomes. For example, in [35], the authors maintain the overall "U" shape using encoder/decoder blocks to replace the classic 3x3 convolution. Similarly, [34], [42] present an U-net that uses residual blocks from ResNet [10].

III. Dynamic Structured Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve programs in an arbitrary language [33]. It evolves solutions encoded as variable-length binary strings that are translated, by means of a mapping process, to executable programs. First, a grammar provided by the user contains a set of rules and components that define a space of possible program designs for a given domain. Then, the grammar is used to decide which components and structures will be used, according to the values in the solution.

We use an improved version of GE, called Dynamic Structured Grammatical Evolution (DSGE) [3], [21]. It was proposed to address and overcome some limitations that the classic GE has. DSGE introduces a new encoding, using a one-to-one relation between genotype and phenotype. The new encoding allows a more efficient way to generate and recombine solutions and also avoids having low locality or redundancy issues.

The GE approach can be defined by three main components: a) a grammar; b) the genotype-phenotype mapping; and c) the search engine. While the grammar and the mapping are used to define and build the programs, the search engine is responsible for searching for new, improved solutions. More details about each component are given in the following sections.

A. Grammar

GE grammar is a set of rules and productions that follows the Backus Naur Form (BNF), which can be used to build the programs.

Formally speaking, a grammar is a tuple $G = (N, T, S, P)$, where N is a non-empty set of non-terminal symbols, T is a non-empty set of terminal symbols, S is an element of N called axiom and used as start rule, and P is a set of productions of the form $A ::= \alpha$, with $A \in N$ and $\alpha \in (N \cup T)^*$, N and T are disjoint [22]. Figure 2 shows an example of a simple grammar, used to form mathematical expressions. In this example, the set N of non-terminals is $\{start, expr, op, term\}$, the set of terminals T is $\{+, -, *, /, x1, 0.5\}$, and the axiom S is given by the non-terminal $start$.

$$\langle start \rangle ::= \langle expr \rangle \langle op \rangle \langle expr \rangle \mid \langle expr \rangle$$

$$\langle expr \rangle ::= \langle term \rangle \langle op \rangle \langle term \rangle \mid (\langle term \rangle \langle op \rangle \langle term \rangle)$$

$$\langle op \rangle ::= \texttt{+} \mid \texttt{-} \mid \texttt{/} \mid \texttt{*}$$

$$\langle term \rangle ::= \text{x1} \mid 0.5$$

Fig. 2. Example of a grammar for mathematical expressions.

B. DSGE Encoding

For DSGE, rather than encoding the solutions as a list of integers, they are encoded as a list of lists, where each internal list is directly tied to one non-terminal symbol from the grammar (see Figure 3). The length of a solution is defined by the number of non-terminals in the grammar. Moreover, the size of each internal list will depend on the number of values needed to perform a complete map, and the values will depend on the number of options for each rule. For example, the non-terminal <op> has four options $(+, -, *, /)$, so the possible values range from 0 to 3.

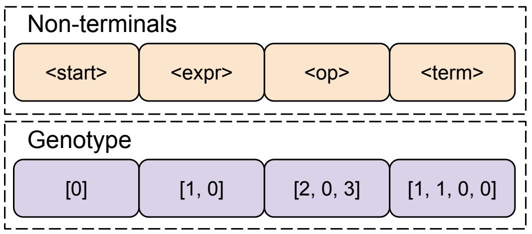

Fig. 3. DSGE encoding. Each list of integers in the genotype is related to a non-terminal.

C. Mapping

The mapping is a procedure that translates one solution into an executable program. It performs a grammar expansion, replacing the leftmost non-terminal by one of its productions, starting from the axiom. In DSGE, instead of picking values sequentially from the genotype as the expansion occurs, the values are selected from the internal list related to the non-terminal being evaluated. Moreover, the values in the solution are within each non-terminal boundaries, which removes the need for using the mod operator as in classical GE.

Derivation step	Integers left
<start>	[[0], [1, 0], [2, 0, 3], [1, 1, 0, 0]]
<expr><op><expr>	[[], [1, 0], [2, 0, 3], [1, 1, 0, 0]]
(<term><op><term>)<op><expr>	[[], [0], [2, 0, 3], [1, 1, 0, 0]]
(0.5 <op><expr>)<op><expr>	[[], [0], [2, 0, 3], [1, 0, 0]]
(0.5 / <term>)<op><expr>	[[], [0], [0, 3], [1, 0, 0]]
(0.5 / 0.5)<op><expr>	[[], [0], [0, 3], [0, 0]]
(0.5 / 0.5) + <expr>	[[], [0], [3], [0, 0]]
(0.5 / 0.5) + <expr><op><term>	[[], [], [3], [0, 0]]
(0.5 / 0.5) + x1 <op><term>	[[], [], [3], [0]]
(0.5 / 0.5) + x1 * <term>	[[], [], [], [0]]
(0.5 / 0.5) + x1 * x1	[[], [], [], []]

Fig. 4. Mapping process using the DSGE encoding.

For example, the mapping displayed in Figure 4 starts from the axiom <start>. It consumes the first value from the first list, which is related to the non-terminal <start>. The value is 0, which means that the non-terminal will be replaced by the production (according to Grammar 2) <expr><op><exp>. Following, the next non-terminal is <expr>, and the next value related to this non-terminal is 1, which will replace <expr> by the production (<term><op><term>). The mapping procedure repeats these steps until all values are used, and there are no non-terminals left.

D. Search Engine

The search engine is a traditional GA, modified to work with the new encoding. DSGE does not require the specific operators, prune and duplication, used by classical GE to control the length of solutions. The mutation operators remain simple. One random value is selected from the solution to be replaced by a new one, considering the possible values according to the evaluated non-terminal. For the crossover operator, the "cut" point is chosen among the non-terminals, and two new solutions are generated by combining these parts. At this point, it might can occur that the newly generated solutions have more or fewer values than necessary for the mapping. In these cases, we repair the solution by adding random values as necessary or removing unused values. Additional details about the DSGE operators can be found in [3].

IV. Grammatical Evolution for Segmentation networks

Our approach is motivated on proposing an efficient way to design segmentation networks following the U-net structure. We developed a grammar that combines an easy way of defining which components can be used to design the networks, with the addition of structural rules that guide the designs towards valid architectures, minimizing the generation of invalid models.

A. U-net mirror grammar

Our goal is to specify the process of constructing an U-net using a grammar. The grammar should be able to generate architectures that fulfill the specific structural characteristics of U-nets explained in Section II-B.

However, while satisfying the main architectural constraints of U-nets, we also want to allow the generation of a variety of U-net designs. Finding the right balance between faithfulness to the original design and ability to explore new designs is a difficult objective. For example, the original architecture applies a concatenate layer on every "level". Therefore we propose sets of rules that both ensure the U-net constraints on designed networks and, at the same time, allow novelty in terms of generating different structures. For example, the grammar can include a rule that specifies whether or not to add a concatenate layer, or in more complex cases, to add skip-connections or robust modules.

B. Encoding U-net constraints

We assume that the U-net architecture has a perfectly symmetric structure, and we introduce a "mirror" grammar that exploits the symmetries for representing the characteristics of only the contracting path of the U-net. The assumption here is that the encoding of the contracting path is sufficient to reconstruct the expanding path of the network. Moreover, by designing only half of the network, we deal with only half of the complexity, which makes the search easier.

```
⟨unet⟩ ::= ⟨conv⟩ ⟨next⟩

⟨next⟩ ::= ⟨conv⟩ ⟨next⟩
     | ⟨dropout⟩ ⟨next⟩
     | ⟨pool⟩ ⟨nextp⟩
     | bridge ⟨pool⟩ ⟨nextp⟩

⟨nextp⟩ ::= ⟨conv⟩ ⟨next⟩ | ⟨middle⟩

⟨middle⟩ ::= ⟨middle⟩ ⟨middle⟩ | ⟨conv⟩ | ⟨dropout⟩

⟨conv⟩ ::= conv ⟨filters⟩ ⟨k_size⟩ ⟨strides⟩ ⟨padding⟩ ⟨activ⟩

⟨pool⟩ ::= ⟨p_type⟩ ⟨strides⟩ ⟨padding⟩ ⟨padding⟩

⟨dropout⟩ ::= dropout ⟨rate⟩

⟨p_type⟩ ::= maxpool | avgpool

⟨filters⟩ ::= 2 | 4 | 8 | ... | 64

⟨k_size⟩ ::= 1 | 2 | ... | 6

⟨strides⟩ ::= 1 | 2

⟨padding⟩ ::= valid | same

⟨activ⟩ ::= relu | sigmoid | linear

⟨rate⟩ ::= [0.0, 0.5]
```

Fig. 5. U-net mirror grammar

Figure 5 presents our proposed Grammar. It includes the main components of a U-net: the convolutional, pooling, and dropout layers, as well as their parameters: for instance, the number of filters, kernel size, and padding. In addition to that, to reproduce the "U" shape on the network, we have structural rules (<next>, <nexp>, <middle>), that are used to build the contracting (left) and transition (middle) parts of the network.

Figure 6 shows the steps taken to build the network. The mapping is responsible for reading the pre-built structure generated from the grammar expansion. Then, for each node on the contracting path, starting from the end, new blocks are added to the expanding path, taking into account the type of node. If it is a convolutional layer, a copy of it is added to the expanding path. For pooling layers, an up-sample and a 2x2 convolutional layer are added. Also, if the pooling rule contains the "bridge" keyword, then a concatenate layer is added after the 2x2 convolutional layer. In the end, the input, classification (2x2 convolutional) and output (1x1 convolutional with sigmoid activation) layers are added, since these are common to every architecture. The classification and output layers are responsible to classify each pixel to a value

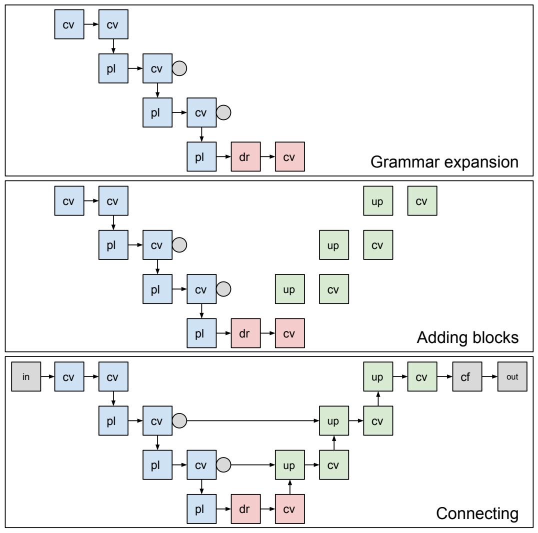

Fig. 6. Representation of the building process using the mirror grammar.

between 0 and 1, and produce the output segmented image. After the build, a repair function is used to ensure that the connections between the nodes are correct. For instance, if a network has a pooling layer that tries to reduce a 1x1 image, nothing will happen; the same way, the correspondent up-sampling should not increase the image size. Otherwise, the image size throughout the network would not match.

This building process proposes an easy way to design the networks while also minimizing the number of invalid models. On the other hand, assuming that all architectures will follow this mirrored philosophy, the grammar will not be able to reproduce the same architecture as the original U-net, since it is not symmetric. Part of the flexibility is sacrificed in order to maintain the balance between generating valid solutions and being able to generate a wider variety of models.

C. Evaluation

The evaluation of the architectures in our approach considers two different scenarios. In both scenarios, the U-net architecture receives two sets of images (train and test) and produces an output for the *test* dataset. In the first (no-training) scenario, the train dataset is ignored, and the parameters of the network are randomly set. In the second (training) scenario, the train dataset is used to learn the parameters of the network.

The evaluation step is the most resource and time-consuming part of the approach when we have to consider the time needed to train and test a network. Many factors contribute to increasing the evaluation time, for instance, the number of epochs, size of the dataset, and overall complexity of a solution. Some of the techniques that can be used to optimize the execution time include using an early stop or a time limit. The first halts the execution if the network does not improve its accuracy within a given number of epochs. The latter imposes a maximum time for the network to be trained. We combine these techniques with our proposed grammar, which minimizes the complexity to design the networks, and the two scenarios mentioned previously, providing better control over the available resources.

1) Image segmentation metrics: To measure the accuracy of the networks, we use a metric that is composed of different coefficients, such as Jaccard, Dice, Sensitivity, and Specificity. As a comparison, in the U-Net original work [32], they only use the Intersection over Union (IoU), which is the same as the Jaccard.

Considering A and B as two images, the Jaccard coefficient (Jacc, Eq. 1), also known as Intersection over Union, is a statistic used for gauging the similarity and diversity of sample sets. The Dice coefficient (DCS, Eq. 2) is similar to the Jaccard but more popular in image segmentation tasks. We also use the well known Sensitivity and Specificity metrics.

$$Jacc = \frac{A \cap B}{A \cup B} \quad (1) \qquad DCS = \frac{2(A \cap B)}{A + B} \quad (2)$$

The accuracy for a model is calculated using a weighted mean, giving the same weight for each of the four measures. The inverse measure is used as the loss function.

V. Experiments

The main goal of our experiments is to evaluate the proposed GE approach in exploring a wider variety of U-net architectures, identifying models that are efficient for image segmentation. This section is organized in the following way. First, we explain the image segmentation benchmark used to evaluate the algorithms. Then, we present the parameters used by the GE approach. Finally, we present the numerical results of the experiments and discuss them.

A. Datasets with varying levels of difficulty

We created an artificial benchmark where, for each image, we have its groundtruth segmentation. The benchmark comprises 5 datasets having increasing levels of difficulty. The original images from which new segmentated images were generated were taken from the texture dataset cited in [1]. The datasets were named as “simple”, “regular”, “moderate”, “hard”, and “full”. Each dataset contains images (Figure 7) of 256x256 pixels, being 80 images for training, 20 for validation and 20 for test. Each image was generated having an original texture image as the background. To this background, a random number of ellipses, which vary in size and quantity, were added. Each ellipse is filled with the pixels of another texture image. The difficulty of each dataset is determined by characteristics such as, the number and size of the ellipses, the contrast between background and foreground textures, and so on. The configuration of each datasets is presented in Table I. For the Regular dataset, if the background selects a dark image, the foreground will be a light image and vice versa.

B. Parameters of the algorithm

The parameters used for the experiments are presented in Table II. We investigated our approach in two scenarios: with and without training of the networks. When the neural network

TABLE I
CONFIGURATION OF EACH DATASET

Dataset	Background	Foreground	Ellipses	Size
Simple	fixed	fixed	[1, 10]	[10, 50]
Regular	light or dark*	dark or light*	[1, 10]	[10, 50]
Moderate	light	dark	[1, 10]	[10, 50]
Hard	fixed	all	[1, 10]	[10, 50]
Full	all	all	[1, 10]	[10, 50]

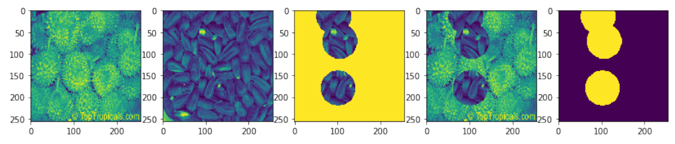

Fig. 7. Example of the dataset generator. From left to right: background image, foreground image, random ellipses, generated image, and segmentation.

is not trained, the parameters it uses are randomly generated. The population size is 20, initialized randomly, evolved for a maximum of 10 generations for the training scenario, limited due to time, and 50 generations for the no-training. Each network, on the training scenario, is trained for 10 epochs, and a one hour of time limit as a heuristic criterion, after which the algorithm is stopped even if the maximum number of epochs has not been reached. The fitness of the solutions is calculated over a validation set. Here, only the best results are reported.

TABLE II
PARAMETERS OF THE EVOLUTIONARY ALGORITHM

Parameter	Value	Parameter	Value
Population size	20	Generations	10/50
Crossover	0.9	Epochs	10
Mutation	0.01	Model time limit	60 min

Regarding the grammar, we also constrained some rules in order to increase the amount of valid architectures and also decrease the overall complexity, that directly affects the time needed to evaluate the solutions. The modifications include fixing the strides to 1 and padding to "same" for the <conv> rule. Use only "maxpool" with strides 2, pool size of 2 and padding "same". And lastly, constraining the number of filters to a max of 32. These modifications allowed our approach to generate 100% of valid U-nets, meaning the evolution will deal with a broader variety of possible solutions instead of ignoring invalid ones. Other combinations of rules and parameters will be further explained in future works.

C. Results

The first question we address is whether the evolutionary process has any impact on the quality of the solutions. Figures 8 and 9 display the mean accuracy per generation for all five datasets with and without training, respectively. In both scenarios, there were clear improvements in the quality of the solutions over the generations. In the training scenario, the U-nets were able to achieve higher accuracy values in early generations, maintaining a slower pace around generation 3. Similarly, in the no-training scenario, solutions are constantly improving, however with small increments on the quality of solutions.

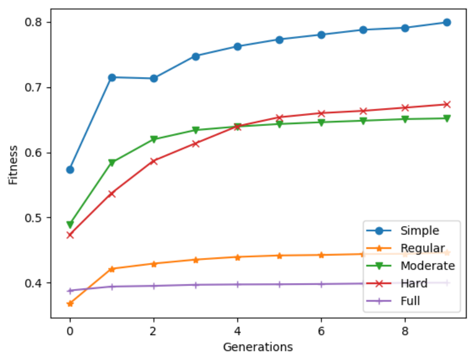

Fig. 8. Mean accuracy per generation during the evolution with training.

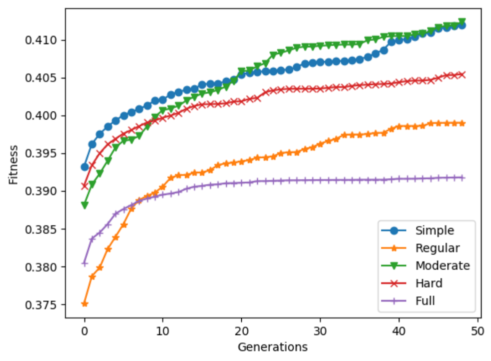

Fig. 9. Mean accuracy per generation during evolution with no training.

Further, we analyzed the relationship between the fitness of the u-nets and their complexity in terms of the parameters. Figures 10 and 11 shows the relation of the average fitness and complexity per generation for scenarios with training and no training respectively. In the figures, there is a chart for each database, and the intensity of the color means the agglomeration of different networks with similar number of parameters. It can be seen in Figures 10 and 11 that for the training scenario, most solutions reported an average of forty thousand (40000) parameters, while on the no-training scenario, the biggest networks have an average of fifty thousand (50000) on the dataset "Full", and other datasets have even smaller values ranging from ten to thirty five thousand. As a comparison value, the original U-net architecture we used has over thirty-one million parameters (31,031,685). The results indicate that the complexity of the networks do not depend on the difficulty of the dataset since complex networks are also evolved for the simplest dataset. However, U-Nets of similar complexity can produce predictions of different quality.

In Table III, we present the accuracies obtained by the best models generated by DSGE for each dataset. The Pre-trained models are the ones obtained from the training scenario, meaning the reported accuracy is the one achieved during the evolution. On the other hand, the Pos-trained models were obtained from the no-training scenario, and the reported

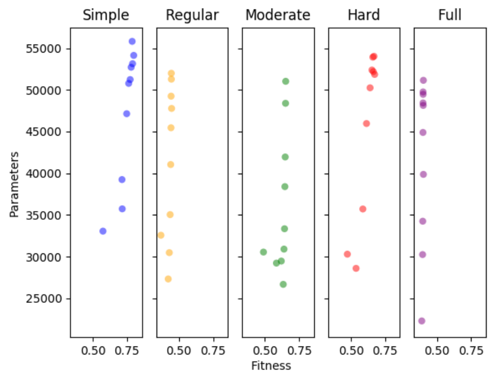

Fig. 10. Average fitness x parameters per generation with training.

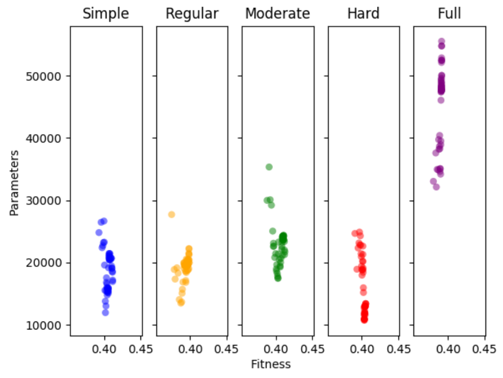

Fig. 11. Average fitness x parameters per generation with no training.

accuracy was obtained after training for 10 epochs. Moreover, the U-net was also trained for 10 epochs for each dataset.

In general, the pre-trained models were able to achieve higher accuracy values than both the pos-trained and the original U-net, with an exception for the dataset "Simple", where the U-net achieved the highest accuracy. Considering that the designed models have a considerably smaller complexity, the fact that the pre-trained models were able to achieve better results than the original U-net, indicates that the obtained architecture was better adapted for the dataset.

TABLE III
ACCURACY OF GENERATED MODELS AND ORIGINAL U-NET

Model	Simple	Moderate	Regular	Hard	Full
Pre-trained	0.9611	**0.6763**	**0.6737**	**0.7828**	**0.4388**
Pos-trained	0.6484	0.3936	0.5899	0.5094	0.3936
Orig. U-net	**0.9779**	0.3596	0.3665	0.3588	0.3624

The performed experiments showed that it is possible to design and improve the performance of a population of networks over time, either by training or not, the networks during the process. Even though the untrained models did not achieve competitive results compared to the trained models, they can be useful as a starting point for evolution conducted using training. Furthermore, in comparison to the original U-net architecture, our approach was able to generate considerably smaller and less complex architectures, that were still able to reach competitive, and even better results for some of the datasets we used.

VI. Conclusion

While several neuroevolutionary approaches have been proposed for CNNs, this is the first work that addresses the question of evolving U-nets, a class of CNNs with constraints in the architecture that makes it harder to solve with sequential representations.

We have introduced a mirror grammar which efficiently represents the U-net constraints and it is flexible enough to represent a wider variety of architectures. We have shown how to use DSGE with this grammar to evolve the U-nets.

We tested the capability of our approach in generating networks within two scenarios, with and without training. The results showed that it is possible to generate designs that improve over time. Our proposed grammar generated a variety of very small and less complex architectures, compared to the original U-net. Moreover, we compared the best network for each dataset in each scenario with the original U-net, and the models trained during the evolution achieved better results in most datasets, when compared to the U-net. One limitation of our analysis is that, due to the computational cost of the evaluation, only few runs of each configuration were run. This could be addressed in the future by considering ways to decrease the evaluation cost and/or increasing the computational resources available.

A. *Future work*

This work can be further expanded in a number of ways, here are some worth mentioning. Improving the grammar representation in order to remove the restrictions and increase the possibilities on generating the networks. Further explore the possibility of generating designs without training to be used as a base for further manual improvements. Explore the viability of mixing characteristics of networks, for instance, adding sets of building blocks to the grammar. Find a more efficient way to evaluate the generated designs and decrease the bottleneck related to computational cost. Another direction worth of research, is conducting detailed comparison to classical techniques for image segmentation.

Acknowledgments

This work was partially supported by Brazilian Education Ministry – CAPES and Brazilian Research Council – CNPq.

A. Mendiburu and R. Santana acknowledge support by the Spanish Ministry of Science, Innovation and Universities (Project TIN2016-78365-R), and the Basque Government (IT1244-19 and ELKARTEK Programs).

References

[1] N. Ahuja and S. Todorovic, "Extracting Texels in 2.1D Natural Textures," in *Proceedings of the 2007 IEEE International Conference on Computer Vision (ICCV'07)*. IEEE, 2007, pp. 1–8.

[2] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Towards the Evolution of Multi-Layered Neural Networks: A Dynamic Structured Grammatical Evolution Approach," in *Proceedings of the 2017 Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 2017, pp. 393–400.

[3] ——, "DENSER: deep evolutionary network structured representation," *Genetic Programming and Evolvable Machines*, vol. 20, no. 1, pp. 5–35, 2019.

[4] L. C. T. Bezerra, M. López-Ibánez, and T. Stützle, "Automatic Component-Wise Design of Multiobjective Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 403–417, June 2016.

[5] J. Cheng, L. Dong, and M. Lapata, "Long Short-Term Memory-Networks for Machine Reading," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP'16)*, pp. 551–561, November 2016.

[6] R. Eberhart and J. Kennedy, "A New Optimizer Using Particle Swarm Theory," in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS'95)*. IEEE, October 1995, pp. 39–43.

[7] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, March 2008.

[8] A. Graves, A. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13)*. IEEE, May 2013, pp. 6645–6649.

[9] J. T. Guibas, T. S. Virdi, and P. S. Li, "Synthetic Medical Images from Dual Generative Adversarial Networks," *arXiv preprint arXiv:1709.01872*, 2017.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the 2016 IEEE conference on Computer Vision and Pattern Recognition (CVPR'16)*, 2016, pp. 770–778.

[11] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An Automatic Algorithm Configuration Framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, October 2009.

[12] F. Hutter, H. H. Hoos, and T. Stützle, "Automatic Algorithm Configuration based on Local Search," in *Association for the Advancement of Artificial Intelligence (AAAI'07)*, vol. 7, 2007, pp. 1152–1157.

[13] Y. Kassahun and G. Sommer, "Efficient Reinforcement Learning Through Evolutionary Acquisition of Neural Topologies." in *Proceedings of the 2005 European Symposium on Artificial Neural Networks (ESANN'05)*, 2005, pp. 259–266.

[14] J. R. Koza, *Genetic programming II: Automatic discovery of reusable subprograms*. The MIT Press, 1994, vol. 13, no. 8.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[16] A. Kumar, O. N. Murthy, P. Ghosal, A. Mukherjee, D. Nandi *et al.*, "A Dense U-Net Architecture for Multiple Sclerosis Lesion Segmentation," in *Proceedings of the 2019 IEEE Region 10 Conference (TENCON'19)*. IEEE, October 2019, pp. 662–667.

[17] R. H. R. Lima and A. Pozo, "A Study on Auto-Configuration of Multi-Objective Particle Swarm Optimization Algorithm," in *Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC'17)*. IEEE, June 2017, pp. 718–725.

[18] ——, "Evolving Convolutional Neural Networks through Grammatical Evolution," in *Proceedings of the 2019 Genetic and Evolutionary Computation Conference (GECCO'19)*. ACM, 2019, pp. 179–180.

[19] R. H. R. Lima, A. Pozo, and R. Santana, "Automatic design of convolutional neural networks using grammatical evolution," in *2019 8th Brazilian Conference on Intelligent Systems (BRACIS'19)*. IEEE, 2019, pp. 329–334.

[20] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The IRACE package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[21] N. Lourenço, F. Assunção, F. B. Pereira, E. Costa, and P. Machado, "Structured Grammatical Evolution: A Dynamic Approach," in *Handbook of Grammatical Evolution*. Springer, 2018, pp. 137–161.

[22] N. Lourenço, F. Pereira, and E. Costa, "Evolving Evolutionary Algorithms," in *Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC'12)*. ACM, 2012, pp. 51–58.

[23] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing Neural Networks using Genetic Algorithms," in *Proceedings of the International Conference on Genetic Algorithms (ICGA'89)*, vol. 89, 1989, pp. 379–384.

[24] J. F. Miller and P. Thomson, "Cartesian Genetic Programming," in *Proceedings of the European Conference on Genetic Programming (EuroGP'00)*. Springer, 2000, pp. 121–132.

[25] P. B. Miranda and R. B. Prudêncio, "GEFPSO: A Framework for PSO Optimization based on Grammatical Evolution," in *Proceedings of the 2015 Genetic and Evolutionary Computation Conference (GECCO'15)*. ACM, 2015, pp. 1087–1094.

[26] P. Mirunalini, C. Aravindan, A. T. Nambi, S. Poorvaja, and V. P. Priya, "Segmentation of Coronary Arteries from CTA axial slices using Deep Learning techniques," in *Proceedings of the 2019 IEEE Region 10 Conference (TENCON'19)*. IEEE, October 2019, pp. 2074–2080.

[27] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski, "Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*, 2017, pp. 4467–4477.

[28] M. O'Neill and C. Ryan, "Grammatical Evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[29] R. Poli, C. Di Chio, and W. B. Langdon, "Exploring Extended Particle Swarms: A Genetic Programming Approach," in *Proceedings of the 2005 Genetic and Evolutionary Computation Conference (GECCO'05)*. ACM, 2005, pp. 169–176.

[30] R. Poli and J. Koza, "Genetic Programming," in *Search Methodologies*. Springer, 2014, pp. 143–185.

[31] M. Rocha, P. Cortez, and J. Neves, "Evolution of Neural Networks for Classification and Regression," *Neurocomputing*, vol. 70, no. 16-18, pp. 2809–2816, 2007.

[32] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proceedings of the 2015 International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'15)*. Springer, 2015, pp. 234–241.

[33] C. Ryan, J. J. Collins, and M. O'Neill, "Grammatical Evolution: Evolving Programs for an Arbitrary Language," in *Proceedings of the European Conference on Genetic Programming (EuroGP'98)*. Springer, 1998, pp. 83–96.

[34] D. Sabarinathan, M. P. Beham, and S. Roomi, "Hyper Vision Net: Kidney Tumor Segmentation Using Coordinate Convolutional Layer and Attention Unit," *arXiv preprint arXiv:1908.03339*, 2019.

[35] M. Salem, S. Valverde, M. Cabezas, D. Pareto, A. Oliver, J. Salvi, À. Rovira, and X. Lladó, "Multiple Sclerosis Lesion Synthesis in MRI using an Encoder-Decoder U-Net," *IEEE Access*, vol. 7, pp. 25 171–25 184, 2019.

[36] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks Through Augmenting Topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[37] M. Suganuma, S. Shirakawa, and T. Nagao, "A Genetic Programming Approach to Designing Convolutional Neural Network Architectures," in *Proceedings of the 2017 Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 2017, pp. 497–504.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*, June 2015.

[39] I. Tsoulos, D. Gavrilis, and E. Glavas, "Neural Network Construction and Training using Grammatical Evolution," *Neurocomputing*, vol. 72, no. 1-3, pp. 269–277, 2008.

[40] B. Wang, Z. Chen, W. Dewulf, R. Pauwels, Z. Yao, Q. Hou, and Y. Xiao, "U-Net-based Blocked Artifacts Removal Method for Dynamic Computed Tomography," *Applied Optics*, vol. 58, no. 14, pp. 3748–3753, 2019.

[41] D. Whitley, T. Starkweather, and C. Bogart, "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity," *Parallel Computing*, vol. 14, no. 3, pp. 347–361, 1990.

[42] Z. Zhang, Q. Liu, and Y. Wang, "Road Extraction by Deep Residual U-Net," *Proceedings of the 2018 IEEE Geoscience and Remote Sensing Letters (GRSL'18)*, vol. 15, no. 5, pp. 749–753, 2018.