Evolving the Topology of Large Scale Deep Neural Networks

Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro

CISUC, Department of Informatics Engineering,
University of Coimbra, Coimbra, Portugal
{fga, naml, machado, bribeiro}@dei.uc.pt

Abstract. In the recent years Deep Learning has attracted a lot of attention due to its success in difficult tasks such as image recognition and computer vision. Most of the success in these tasks is merit of Convolutional Neural Networks (CNNs), which allow the automatic construction of features. However, designing such networks is not an easy task, which requires expertise and insight. In this paper we introduce DENSER, a novel representation for the evolution of deep neural networks. In concrete we adapt ideas from Genetic Algorithms (GAs) and Grammatical Evolution (GE) to enable the evolution of sequences of layers and their parameters. We test our approach in the well-known image classification CIFAR-10 dataset. The results show that our method: (i) outperforms previous evolutionary approaches to the generations of CNNs; (ii) is able to create CNNs that have state-of-the-art performance while using less prior knowledge (iii) evolves CNNs with novel topologies, unlikely to be designed by hand. For instance, the best performing CNN obtained during evolution has an unexpected structure using six consecutive dense layers. On the CIFAR-10 the best model reports an average error of 5.87% on test data.

Keywords: Convolutional Neural Networks, Deep Neural Networks, Genetic Algorithm, Dynamic Structured Grammatical Evolution

1 Introduction

Machine Learning (ML) enables machines to learn from large volumes of data, where often there is the need to pre-process the data in order to extract features. To do that, it is required expert knowledge about the problem domain, and then we have to manually design a model that can learn the data patterns. Deep Learning (DL) avoids this by building models that are aimed at learning a representation of the data, thus reducing the amount of required domain knowledge. But, DL models tend to require deep Artificial Neural Networks (ANNs) so that the learning of the problem features is effective. Nonetheless, DL has been successfully applied in many domains, such as, computer vision [23, 7, 18], speech recognition [6, 11], or machine translation [33].

An example of a deep network, which is often used for object recognition is VGG, introduced by Simonyan and Zisserman in [25]. VGG is a 16 to 19 deep

Convolutional Neural Network (CNN), which has pushed the boundaries on the ImageNet Challenge 2014. CNNs, as other DL models, involve a large number of design choices. For instance, one needs to decide on the number of layers, the type of layers, and the parameterisation of the multiple receptive fields that compose it such as the number of filters, stride, or filter sizes. Everyday these models get more and more complex and optimising all the involved parameters is becoming an increasingly arduous task. For that reason, researchers have focused their efforts on automating the design of deep networks. The current work is a step forward in this line of research.

In this article in propose DENSER, a novel representation that is capable of searching for adequate topologies (and the learning hyper-parameters) of CNNs. Although in the current paper we only apply it to the evolution of CNNs, we argue that the proposed method can be applied to different network structures. By combining the principles of a standard Genetic Algorithm (GA) with Grammatical Evolution (GE) [20] we allow the direct evolution of a sequential list of layers, where the parameter values of each layer are encapsulated in a position of the GA genotype, facilitating the application of the genetic operators. In this way we can reuse the method for different network structures and domains, as we only need to change the underlying grammar. We test the proposed approach on an image classification dataset, namely the CIFAR-10 dataset. The results reveal that our method is able to find competitive CNNs, often superior to others reported in the literature. In concrete, the CNN that obtains the best performance on training data, has an accuracy of 94.13%, i.e., an error of 5.87% on test data.

The remainder of the document is organised as follows. In Section 2 we survey NeuroEvolution (NE) works, with special focus on those targeting the evolution of deep structures. Next, in Section 3, we introduce our approach, detailing how we combine the principles of GAs with GE. Experimental results are reported in Section 4. To end, in Section 5 conclusions are drawn, and future work and open questions are addressed.

2 State of the Art

When designing learning models, an exhaustive trial-and-error process is often followed in an attempt to discover which is the configuration that performs best. In particular, and focusing our attention on ANNs (shallow or deep), decisions have to be made considering the topology and weights / learning parameters of the networks. For that reason, the approaches that try to automatically tune the networks are grouped according to the aspects of the network they try to optimise: (i) learning; (ii) structure; (iii) learning and topology.

Several iterative, non-evolutionary approaches, have already been successfully applied to the optimisation of ANNs (e.g., [8]). In the vast majority of these methods only a solution is being optimised, and consequently it is likely that the search procedure will become trapped in local optima. In addition, the aim is often to find the simplest solution; however, the simplest solution is

not necessarily the one that performs best, or is even the one that is easiest to train [2].

The use of Evolutionary Computation (EC) techniques to optimise ANNs defines NeuroEvolution (NE). In NE, the population of candidate solutions that is evolved throughout generations represents ANNs for solving a specific task. The quality of the candidate solutions is measured on how well the encoded networks perform when solving the problem.

The application of Evolutionary Algorithms (EAs) to the optimisation of the learning of fixed network topologies can happen at different levels. The most simple strategy consists of the the use of EAs to optimise the hyper-parameters. Examples of such approaches are described by Kim et al. [13], and Parra et al. [22], focusing on the optimisation of the multiple parameters of the Back-Propagation (BP) algorithm (and its variants). An alternative to evolving the hyper-parameters consists on evolving the actual learning rules that are used for updating the synaptic weights [24]. A particular example of such approaches are those based on the evolution of composition pattern producing functions, i.e., functions that given the position of two neurons in a grid are able to generate the weight associated with that connection [28, 3].

If on the one hand, optimising the parameters of the learning algorithms is a difficult task, it is also true that the majority of the learning algorithms have a gradient-descent nature, and as such, are susceptible to become trapped in a local optimum. Using a population-based search heuristic (such as EAs) is a way to minimise the impact of this issue; to that end, the weights and bias values of ANNs can be directly evolved.

There are various NE works on the search for the appropriate weights and bias values. Usually, the values of the weights are encoded linearly, i.e., a linear sequence of bits [31] or real-values [4], each representing a specific connection; or using a matrix representation [12]. It is also important to mention approaches specifically designed for tuning the weights of ANNs, such as Cooperative Synapse NeuroEvolution (CoSyNE) [9], as well as those that more generally aim at optimising real-values, e.g., G3PCX [5].

When focusing on the development of NE approaches for training ANNs the topology of the networks is often fixed. Notwithstanding, as previously stated, defining the topology is also a laborious process, which requires domain expertise and multiple attempts. The NE methods that tune the structure of networks can be partitioned into three groups, according to how they address the optimisation task: (i) connection; (ii) node; or (iii) layer-based.

In connection-based encodings, the majority of the approaches optimise the connections that are used in a large, a-priori, defined network [15, 21]. However, this limits the search space, disabling the exploration of alternative network structures that are not considered in the pre-defined network. On the other hand, node-based approaches have as base-unit of evolution each single neuron and the connections from and to that neuron. Consequently, they are the most flexible type of approach in what respects the exploration of the search space, as they allow the creation of any sort of structure or node network. Examples of well-known

node-based approaches are: EPNET [32], Symbiotic, Adaptive Neuro-Evolution (SANE) [19], or NeuroEvolution of Augmenting Topologies (NEAT) [29].

Although node-based approaches make the search less restricted and unbiased, they also make it more difficult to search for deep networks, which can be made of thousands or even millions of nodes. That is the reason why the majority of the works focusing on the generation of deep structures use the layers as base unit of evolution. Coevolution DeepNEAT (CoDeepNEAT) [16] combines the ideas behind SANE and NEAT for the evolution of deep networks, where two populations of modules and blueprints are evolved simultaneously. Following the same line of research, in CGP-NN [30] Cartesian Genetic Programming is used in the evolution of the architecture of CNNs. However, instead of promoting the automatic discovery of the most appropriate modules, they are defined a-priori, and only their combination and placement is evolved.

3 Proposed Approach

To promote the evolution of the structure and parameters of the ANNs we propose a novel representation, called DENSER (Deep Evolutionary Network StructurEd Representation), that combines the basic principles of GAs with Dynamic Structured Grammatical Evolution (DSGE) [1]. The sequence of layers is encoded using the GA, and the parameterisation of each layer using DSGE. By doing this, we are able to evolve networks where the genetic material of each layer is kept together, and therefore the manipulation of the solutions is easier, since there is a one-to-one mapping between the layers and their parameters.

In the upcoming sub-sections we further detail the representation used, the genetic operators, and how the generated networks are evaluated, respectively in Sections 3.1, 3.2 and 3.3.

3.1 Representation

Each candidate solution encodes the structure of a single ANN by means of an ordered linear structure, where each position is a functional unit of the network, i.e., a layer. It is also possible to evolve the learning algorithm that should be used to train the network and its hyper-parameters. The motivation to promote a layer-based evolution rather than node or connection-based is related with the desire to tackle challenging problems, which often require deep networks. Such structures have a large number of neurons and connections, which makes their optimisation using a low level representation hard to accomplish.

To facilitate the application of the approach to different network structures and layer types we encode each layer similarly to DSGE, meaning that evolution acts on grammatical derivations. As a results the genotype of each position of the GA (which represents a layer) is encoded as a list of genes, each of them responsible for keeping the expansion possibilities for specific non-terminal symbols of the grammar. In addition to the standard DSGE genotype we introduce a special coding block to deal with integer and float values. This block is represented

```
<features> ::= <convolution>
             | <pooling>
<convolution> ::= layer:conv [num-filters,int,1,32,256] [filter-shape,int,1,1,5]
                  [stride,int,1,1,3] <padding> <activation> <bias>
                  <batch-normalisation> <merge-input>
<batch-normalisation> ::= batch-normalisation:True
                        | batch-normalisation:False
<merge-input> ::= merge-input:True
                | merge-input:False
<pooling> ::= <pool-type> [kernel-size,int,1,1,5] [stride,int,1,1,3] <padding>
<pool-type> ::= layer:pool-avg
              | layer:pool-max
<padding> ::= padding:same
            | padding:valid
<classification> ::= <fully-connected>
<fully-connected> ::= layer:fc <activation> [num-units,int,1,128,2048 <bias>
<activation> ::= act:linear
               | act:relu
               | act:sigmoid
<bias> ::= bias:True
         | bias:False
<softmax> ::= layer:fc act:softmax num-units:10 bias:True
<learning> ::= learning:gradient-descent [lr,float,1,0.0001,0.1]
```

Fig. 1: Example grammar for the encoding of CNNs.

in the grammar in the form of [variable name, variable type, number of values, minimum value, maximum value]. An example of their use in a grammar can be found in Figure 1. At the genotypic level, the block values are kept together with the integers encoding the non-terminal expansion possibilities.

The combination of a GA with DSGE not only makes the approach easily generalisable, but also enables the incorporation of domain knowledge. To define the allowed structure of the networks (i.e., the allowed sequence of layers) the method requires the definition of a list of tuples, where each index of the list indicates the valid grammar starting symbols (for that layer) along with the minimum and maximum number of layers of that type. For example, for searching CNNs the following structure can be specified: [(features, 1, 10), (classification, 1, 2), (softmax, 1, 1), (learning, 1, 1)]. Using the previous example and the grammar of Figure 1, the search space encompasses networks that are

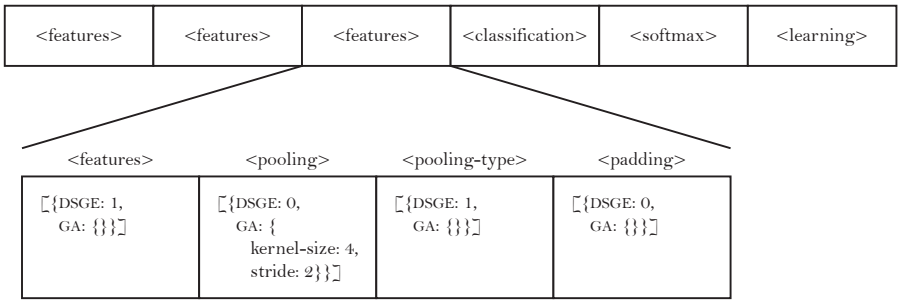

Fig. 2: Example of the genotype of a candidate solution that encodes a CNN.

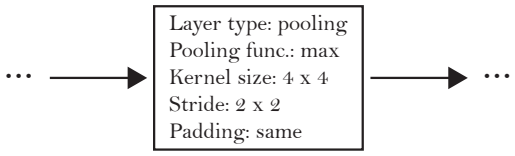

Fig. 3: Phenotype corresponding to the layer specified in Figure 2.

formed by at least one and up to ten convolution or pooling layers (that can be placed in any order). These first convolution and pooling layers are followed by one or two fully-connected ones, and then by an output layer. The output layer is usually encoded as a fully-connected layer with a specific number of neurons corresponding to the number of classes of the problem. On top of the definition of the network topology we also allow the learning parameters to be optimised.

Figure 2 depicts an example of the genotype of a candidate solution, based on the grammar of Figure 1 and on the GA structure introduced above: [(features, 1, 10), (classification, 1, 2), (softmax, 1, 1), (learning, 1, 1)]. As previously explained, the candidate solution has two genotypic levels: (i) the GA level which defines the structure, and points out to the grammar non-terminal symbol that is to be used as the start symbol; and (ii) the DSGE level that stores the ordered sequence of integers encoding the expansion possibilities for each specific non-terminal, and the real-values needed by the networks. Figure 3 presents the phenotype corresponding to the layer which has the DSGE genotype detailed in Figure 2.

3.2 Genetic Operators

To promote the evolution of the candidate solutions we rely on crossover and mutation operators specifically designed for the manipulation of ANNs.

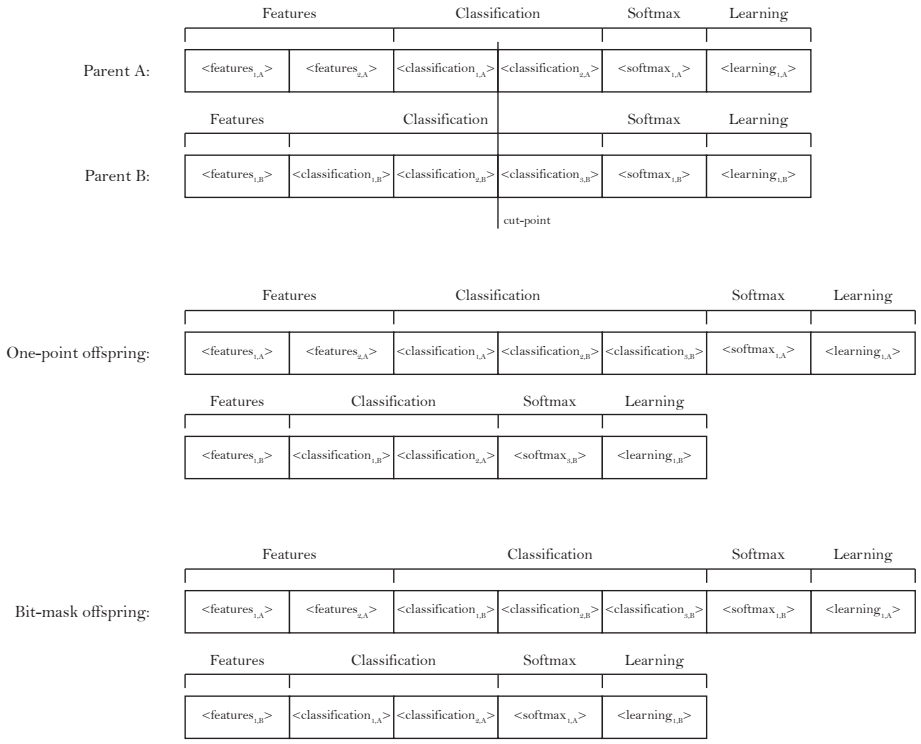

Fig. 4: Example of the introduced crossover operators. The example focuses on the GA level of the genotype. For the bit-mask crossover the mask is 1001, which is associated to the features, classification, softmax and learning modules, respectively.

Crossover

One of the advantages of having two genotypic levels is that the outer level encodes each layer separately. Since the genetic material is encapsulated, devising efficient crossover operators becomes easier. Based on the nature of the genotype, we developed two crossover operators, which are applied probabilistically, each having the same likelihood (i.e., 50%).

Before describing the crossover operators we need to define the notion of module. In this context, the term module does not refer to a set of layers that can be replicated multiple times, but is rather the set of layers that belongs to the same GA structural index. For example, in the above example of a GA structure, the module (features, 1, 10) is composed by all those layers that have their derivation starting with the same non-terminal symbol *features*. The two crossover operators are applied at different levels: one changes layers within a specific layer module, while the other swaps entire modules between individuals.

The first operator is based on the principle that each layer has its genetic material encapsulated. Hence, we designed a crossover operator, that generates

two offspring by crossing the layers that belong to the same module of two parents (chosen by tournament selection). The same module in different parents can have a distinct number of layers; to deal with that the cutting point is randomly generated considering the individual that has less layers in the module.

The other crossover operator is loosely based on the uniform operator for binary representations, and acts upon the modules swapping them between individuals. Figure 4 shows an example of the application of the crossover operators.

Mutation

The operators that work at the GA level aim at manipulating the layers and their parameters. For this purpose we developed the following operators:

Add layer – a new layer is generated at random with the initial symbol for the grammatical derivation being the one of the module where the layer will be placed. This operator can only be applied in modules where the maximum number of layers has not been reached yet;

Replicate layer – similar to the previous mutation operator, but instead of generating a new random layer uses one that is already in the genotype and copies it into another position of the module. This copy is done by reference, which means that if at any given time the layer or some of its parameters are changed, the modifications are propagated to their replicas;

Remove layer – deletes a random layer from a given module. It is only possible to remove a layer if after removal the number of layers in that module is still above the minimum threshold.

The previous operators act only at a macro level, and thus do not change the parameters of the layers. This is accomplished at the DSGE level:

Grammatical mutation – as in standard DSGE, an expansion possibility is replaced by another valid one;

Integer mutation – an integer block is replaced by a new one, where the integers are generated at random, within the allowed range;

Float mutation – similar to the integer mutation, but where instead of randomly generating new values, a Gaussian perturbation is applied.

3.3 Evaluation

The evaluation of the network is divided into two different steps: (i) the mapping from the genotype to the phenotype; and (ii) the training of the generated ANN.

To decode the genotype, the outer level of each candidate solution is traversed linearly. Remember that the outer level (which corresponds to the GA genotype) is where the initial start symbol for expanding the grammatical derivations of the layers is stored. The grammatical genotype is decoded similarly to DSGE: the integers encoding the expansion possibilities of each non-terminal symbol are used only once sequentially. The main difference is that when the expansion of

the non-terminal symbol hits a block that represents an integer or float value the corresponding integer or float value is read from the grammatical GA genotype.

To train the evolved networks we used Keras, running on top of TensorFlow. The dataset used to validate our approach is partitioned into three disjoint sets:

Train – used to train the network using the defined or evolved learning parameters. The parameters vary depending on the used learning algorithm;
Validation – used to evaluate the performance of the network during evolution;
Test – kept aside from the evolutionary process, and used to evaluate the performance of the best models on unseen data, so we can better understand their generalisation ability.

Each network is trained during 10 epochs, and the fitness is the best performance on the validation set on the 10 epochs. Data augmentation is used, namely, padding, horizontal flips, and random crops. A more detailed explanation of the data augmentation approach followed can be found in [30].

4 Experimentation

To test the approach we conducted experiments on the evolution of CNNs for the classification of the CIFAR-10 dataset (further detailed in Section 4.1). The experimental setup used is described in Section 4.2, and the analysis of the experimental results is carried out in Section 4.3.

4.1 Problem Description

The CIFAR-10 dataset [14] is composed of images of 10 disjoint classes, namely: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. For each class there are 6000 cases, making a total of 60000 instances. Each instance is a 32×32 RGB colour image. The goal is to train a CNN that can correctly identify the class of each sample, maximising the accuracy of the object recognition task.

4.2 Experimental Setup

Table 1 shows the parameters used in the experiments. As discussed before, each network is trained during 10 epochs, using the backpropagation learning algorithm, and a learning rate of 0.01. Fitness is measured using the validation set. After the evolutionary cycle, and to further tune the best generated models, we merge the train and validation sets, so that more data is available for training the best topologies found; the networks are trained during 400 epochs with the same learning rate policy. The test data is not changed, and is used to measure the final performance of the best networks found during evolution.

The topology of the evolved networks is constrained to the following GA structure: [(features, 1, 30), (classification, 1, 10), (softmax, 1, 1)], and the experiments are conducted with a grammar similar to the one presented in Figure 1. This way, we allow the evolution of networks that can have up to 40

Table 1: Experimental parameters.

Evolutionary Engine Parameter	**Value**
Number of runs	10
Number of generations	100
Population size	100
Crossover rate	70%
Mutation rate	30%
Tournament size	3
Elite size	1%
Dataset Parameter	**Value**
Train set	42500 instances
Validation set	7500 instances
Test set	10000 instances
Data Augmentation Parameter	**Value**
Padding	4
Random crop	4
Horizontal flipping	50%

hidden-layers: up to 30 convolution or pooling layers followed by at most 10 fully-connected layers. We use the same data augmentation strategy of [30]: each training instance is applied a padding of 4; then we randomly crop the padded image to 32×32, followed by random horizontal flipping.

4.3 Experimental Analysis

Figure 5 depicts the evolution of the average fitness and number of layers of the best CNNs across generations. A brief perusal of the results indicates that evolution is occurring, and solutions tend to converge around the 80th generation. Two different and contradictory behaviours are observable. From the start of evolution and until approximately the 60th generation an increase in performance is accompanied by a decrease in the number of layers; this changes from the 60th generation until the last generation where an increase in performance is followed by an increase in the number of hidden-layers of the best networks. To support this analysis we compute the correlation between the average fitness values of the best individuals and the average number of layers, per generation. The Pearson correlation reports a coefficient of -0.7166 (moderate negative correlation) for the correlation between the two metrics before the 60th generation; after the 60th generation the coefficient is 0.9204 (strong positive correlation).

This analysis reveals an apparent contradiction, that is explained after the fact that in the first generation the randomly generated solutions have a large number of layers (approximately 15.6), which correspond to very deep networks. However, since the numeric parameters of each layer are set at random, they would hardly provide any meaningful results. As evolution proceeds and optimises the numeric values, the best solutions can steadily increase the number of

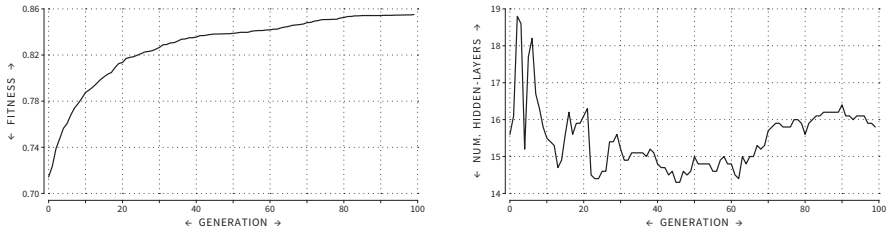

Fig. 5: Evolution of the fitness (left) and number of layers (right) of the best individuals across generations. Results are averages of 10 independent runs.

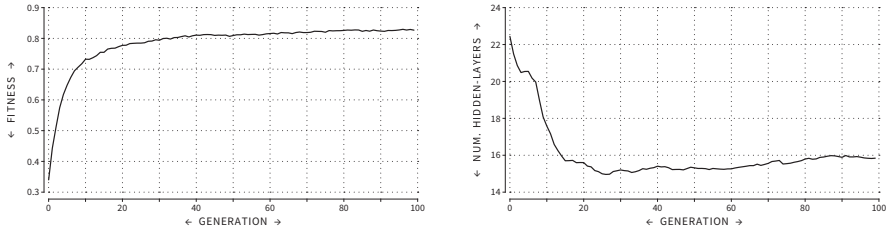

Fig. 6: Evolution of the fitness (left) and number of layers (right) of the overall population across generations. Results are averages of 10 independent runs.

layers to improve their performance. This indicates that it may be advantageous to start the evolutionary process with shallower networks.

In addition to analysing the best evolved solutions we also inspect the overall quality of the population. Figure 6 shows the evolution of the fitness, and number of layers, across generations at the population level. The conclusions are in line with those reported for the analysis of the best solutions, however the change in behaviour occurs earlier, around the 25th generation. Before the 25th generation the Pearson correlation reports a coefficient of -0.89 (strong negative correlation), and after a coefficient of 0.8801 (strong positive correlation). The change in behaviour happens earlier than when considering only the best solutions because in the first generations the population has many low performing solutions that are quickly discarded.

The fittest network found during evolution (in terms of validation accuracy) is represented in Figure 7. As it can be observed, several lambda layers exit. This is due to the fact that the employed grammar allows merging the output of the convolution layers (Conv2D) with the input, using the Add layer. When the number of channels to be merged is different, we pad the one that has less channels, using the Keras Lambda layer. When the signals do not have the same width and height we down-sample the largest one, by applying max pooling.

The most puzzling characteristic of the evolved network is the importance and number of the fully-connected (i.e., dense) layers that are used at the end of the topology. Other approaches on the evolution of CNNs tend to disregard fully-connected layers, and focus only on convolution and pooling layers. We tried to

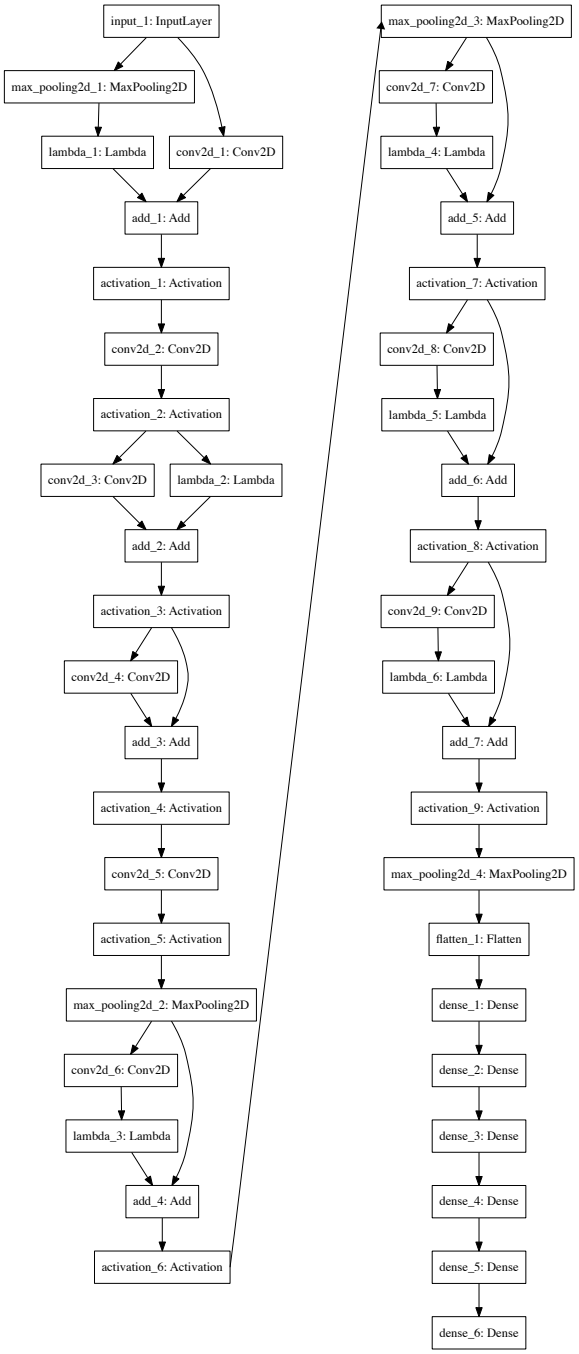

Fig. 7: Topology of the best network during evolution.

remove some of the fully-connected layers, and preliminary results show that the performance of the network degenerated. Moreover, to the best of our knowledge, the sequential use of a such large number of dense layers is unprecedented, and it is fair to say that a human would never think of such topology, which makes this evolutionary outcome remarkable.

Once the evolutionary process is completed, the best network found in each run, i.e. the one obtaining the highest fitness value, is re-trained 5 times, with different initial weights. These networks are selected according to their accuracy on the validation set, to ensure that we have an unbiased selection. The results regarding accuracy reported bellow are averages of these 5 trains for each network.

First, we train the networks with the same learning rate policy used in evolution, but during 400 epochs. With this setup we obtain, on average, a classification accuracy of 88.41% (error of 11.59%) on the test set. To further enhance the accuracy of the networks we adopt the strategy described by Snoek et al. in [26], i.e., for each instance of the test set we generate 100 augmented images. The label assigned by the model is the class that has the maximum average confidence value on the 100 generated augmented images. Following this validation approach the average accuracy on the test set of the best evolved networks increases to 89.93% (10.07% of error).

Although the average accuracy of the fittest models seems low when compared with state of the art approaches, the accuracy of the fittest network is slightly higher: 92.70% (an error of 7.30%). To investigate if it is possible to increase the performance of the fittest networks we re-train them using the same strategy of CGP-CNN [30]. We use a varying learning rate: it starts at 0.01; on the 5th epoch it is increased to 0.1; by the 250th epoch it is decreased to 0.01; and finally at the 375th it is reduced to 0.001. With the previous training policy the average accuracy of the fittest network increases to 93.38%. Finally, this accuracy is further improved if we follow the guidelines from [26], and perform data augmentation on the test data: 94.13%, i.e., an error of 5.87%, which is a highly competitive result.

Obviously, in an ideal scenario, all the training strategies described above would be used during evolution, however it is unfeasible to do so, since it would require immense computational power. Thus, the experimental results indicate that it is possible to obtain competitive results using evolutionary means and that it is possible to do so with limited computational resources, using a low number of training epochs (10) during evolution.

Table 2 shows a comparison with the best results reported by other methods. An analysis of the results shows that DENSER (i.e., our approach) is the one that reports the lowest error. The number of trainable parameters is much higher in our methodology because we allow the placement of fully-connected layers in the evolved CNNs. In addition to the increase in performance, our approach attains these results without any prior knowledge about the domain. Whilst in CGP-CNN the authors have to define fixed modules of layers that are placed and connected by the evolutionary algorithm to form a CNNs, we do not require

Table 2: Comparison of the best results obtained by different methods on the CIFAR-10 dataset. The error rate is measured on the test set. The number of parameters is the number of values that need to be tuned during training.

Method	Error rate	Number of parameters
CoDeepNEAT [29]	7.3%	–
Snoek et al. [26]	6.37%	–
CGP-CNN (ConvSet) [30]	6.75%	1.52×10^6
CGP-CNN (ResSet) [30]	5.98%	1.68×10^6
DENSER	**5.87%**	10.81×10^6

any definition of modules, which implies that the algorithm must discover the appropriate sequence of layers to construct effective networks.

5 Conclusions and Future Work

The definition of the structure and parameterisation of learning models is a hard and time consuming task. This problem is even more pressing when dealing with deep architectures, where the high number of layers makes the tuning task more difficult to accomplish by hand. To this end several evolutionary methods that seek to automatically solve this issue have been proposed.

In this article we combine two evolutionary methods: GAs and GE. With this combination we are able to evolve linear sequences of layers, where each layer is encoded using a grammar-based approach. Consequently the genetic material associated with each layer is encapsulated, making it easier to apply genetic operators to the candidate solutions. The use of a grammar to specify how the layers are encoded makes the approach easily adaptable to other network structures, layer types and domains.

The experimental results confirm the effectiveness of the approach, which outperforms CGP-CNN [30], CoDeepNEAT [29] and the work by Snoek et al. [26], without resorting to prior knowledge. As such, DENSER is, currently, the most successful method for automatic construction of networks in CIFAR-10 dataset. Moreover, its performance is only surpassed by [10, 27, 17], which resort to prior knowledge.

As future work we intend to further test our approach, performing more experiments to ensure the quality and consistency of the results. Moreover, we plan to evaluate the performance on other classification tasks, to assess the generality of the proposed method.

Acknowledgments

This work is partially funded by: Fundação para a Ciência e Tecnologia (FCT), Portugal, under the grant SFRH/BD/114865/2016, and is based upon work from

COST Action CA15140: ImAppNIO, supported by COST (European Cooperation in Science and Technology): www.cost.eu. We would also like to thank NVIDIA for providing us Titan X GPUs.

References

1. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: Towards the evolution of multi-layered neural networks: A dynamic structured grammatical evolution approach. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 393–400. GECCO '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3071178.3071286
2. Blum, A., Rivest, R.L.: Training a 3-node neural network is np-complete. In: Proceedings of the 1st International Conference on Neural Information Processing Systems. pp. 494–501. MIT Press (1988)
3. Buk, Z., Koutník, J., Šnorek, M.: NEAT in HyperNEAT Substituted with Genetic Programming, pp. 243–252. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), https://doi.org/10.1007/978-3-642-04921-7_25
4. David, O.E., Greental, I.: Genetic algorithms for evolving deep neural networks. In: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation. pp. 1451–1452. ACM (2014)
5. Deb, K., Anand, A., Joshi, D.: A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary computation 10(4), 371–395 (2002)
6. Deng, L., Hinton, G., Kingsbury, B.: New types of deep neural network learning for speech recognition and related applications: An overview. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. pp. 8599–8603. IEEE (2013)
7. Farfade, S.S., Saberian, M.J., Li, L.J.: Multi-view face detection using deep convolutional neural networks. In: Proceedings of the 5th ACM on International Conference on Multimedia Retrieval. pp. 643–650. ICMR '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2671188.2749408
8. Franco, L., Jerez, J.M.: Constructive neural networks, vol. 258. Springer (2009)
9. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. Journal of Machine Learning Research 9(May), 937–965 (2008)
10. Graham, B.: Fractional max-pooling. arXiv preprint arXiv:1412.6071 (2014)
11. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 6645–6649 (May 2013)
12. Junyou, B.: Stock price forecasting using pso-trained neural networks. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. pp. 2879–2885. IEEE (2007)
13. Kim, H.B., Jung, S.H., Kim, T.G., Park, K.H.: Fast learning method for backpropagation neural network by evolutionary adaptation of learning rates. Neurocomputing 11(1), 101–106 (1996)
14. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
15. Leung, F.H.F., Lam, H.K., Ling, S.H., Tam, P.K.S.: Tuning of the structure and parameters of a neural network using an improved genetic algorithm. IEEE Transactions on Neural networks 14(1), 79–88 (2003)

16. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Navruzyan, A., Duffy, N., Hodjat, B.: Evolving deep neural networks. arXiv preprint arXiv:1703.00548 (2017)
17. Mishkin, D., Matas, J.: All you need is a good init. arXiv preprint arXiv:1511.06422 (2015)
18. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
19. Moriarty, D.E., Miikkulainen, R.: Forming neural networks through efficient and adaptive coevolution. Evolutionary Computation 5(4), 373–399 (1997)
20. O'Neil, M., Ryan, C.: Grammatical evolution. In: Grammatical Evolution, pp. 33–47. Springer (2003)
21. Palmes, P.P., Hayasaka, T., Usui, S.: Evolution and adaptation of neural networks. In: Neural Networks, 2003. Proceedings of the International Joint Conference on. vol. 1, pp. 478–483. IEEE (2003)
22. Parra, J., Trujillo, L., Melin, P.: Hybrid back-propagation training with evolutionary strategies. Soft Computing 18(8), 1603–1614 (2014)
23. Plis, S.M., Hjelm, D.R., Salakhutdinov, R., Allen, E.A., Bockholt, H.J., Long, J.D., Johnson, H.J., Paulsen, J.S., Turner, J.A., Calhoun, V.D.: Deep learning for neuroimaging: a validation study. Frontiers in neuroscience 8 (2014)
24. Radi, A., Poli, R.: Discovering efficient learning rules for feedforward neural networks using genetic programming. In: Recent advances in intelligent paradigms and applications, pp. 133–159. Springer (2003)
25. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
26. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable Bayesian optimization using deep neural networks. In: International Conference on Machine Learning. pp. 2171–2180 (2015)
27. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
28. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artificial life 15(2), 185–212 (2009)
29. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 10(2), 99–127 (2002)
30. 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. pp. 497–504. GECCO '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3071178.3071229
31. Whitley, D., Starkweather, T., Bogart, C.: Genetic algorithms and neural networks: Optimizing connections and connectivity. Parallel computing 14(3), 347–361 (1990)
32. Yao, X., Liu, Y.: Evolutionary artificial neural networks that learn and generalise well. In: 1996 IEEE International Conference on Neural Networks, Washington, DC, USA, Volume on Plenary, Panel and Special Sessions. pp. 159–164 (1996)
33. Zhang, J., Zong, C.: Deep neural networks in machine translation: An overview. IEEE Intelligent Systems 30(5), 16–25 (2015)