DENSER: Deep Evolutionary Network Structured Representation

Filipe Assunção [1] **Nuno Lourenço** [1] **Penousal Machado** [1] **Bernardete Ribeiro** [1]

Abstract

Deep Evolutionary Network Structured Representation (DENSER) is a novel approach to automatically design Artificial Neural Networks (ANNs) using Evolutionary Computation (EC). The algorithm not only searches for the best network topology (e.g., number of layers, type of layers), but also tunes hyper-parameters, such as, learning parameters or data augmentation parameters. The automatic design is achieved using a representation with two distinct levels, where the outer level encodes the general structure of the network, i.e., the sequence of layers, and the inner level encodes the parameters associated with each layer. The allowed layers and hyper-parameter value ranges are defined by means of a human-readable Context-Free Grammar. DENSER was used to evolve ANNs for two widely used image classification benchmarks obtaining an average accuracy result of up to 94.27% on the CIFAR-10 dataset, and of 78.75% on the CIFAR-100. To the best of our knowledge, our CIFAR-100 results are the highest performing models generated by methods that aim at the automatic design of Convolutional Neural Networks (CNNs), and is amongst the best for manually designed and fine-tuned CNNs .

1. Introduction

The design of Artificial Neural Networks (ANNs) usually requires an arduous and iterative trial-and-error process, where various aspects of the networks have to be considered. Practitioners have to decide on the network topology, the specific parameters of each layer, which learning algorithm should be used and its parameters, and the parameterisation of other criteria like the data augmentation methods. Such decisions require a high level of expertise, and if not performed with care, we might design models that have a poor performance . This task becomes even more difficult considering that the different decisions that have to be made are not independent from one another. One way to to avoid this laborious process it to resort to networks that have already been constructed for a specific task, and have shown a good performance. Nevertheless, these networks tend to be problem specific, and thus, for each different dataset and/or task they might not give the best results. Another war to overcome this challenge is to rely on automatic methods that seek for the design of ANNs.

There are several iterative approaches that seek the structured optimisation of ANNs. Constructive (Frean, 1990; Parekh et al., 2000) methods start from an elementary structure and add nodes or connections until a network structure that is capable of solving the problem emerges. In contrast, Pruning (Reed, 1993; Molchanov et al., 2016) methods start from a complex network structure, and at each iteration remove nodes or connection. . These methods are based on the theory that small networks generalise more easily; however that is not necessarily true (Sietsma & Dow, 1991). Another limitation of theses methods is that often only a single network is being optimised, and consequently, there is a high chance of the search becoming stagnated in a local optima.

To address the problem of automatic design of ANNs, we propose Deep Evolutionary Network Structured Representation (DENSER), a novel approach for the automatic generation of the topology and hyper-parameters needed to build effective ANNs. DENSER is a layer-based method. The evolution of each neuron directly allows for an high degrees of freedom which results in greater control over the generated structures. However, when dealing with big datasets where large scale networks are needed, the involved number of neurons and connections make evolution at such a low level unfeasible.

The main contributions of this work are:

- DENSER, a general framework based on evolutionary principles that automatically searches for the adequate structure and parameterisation of large scale deep networks that can have different layer types (e.g., convolutional, pooling, fully-connected), and goals (e.g., classification, regression);
- An automatically generated Convolutional Neural Net-

*Equal contribution [1]CISUC, Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal. Correspondence to: Filipe Assunção <fga@dei.uc.pt>.

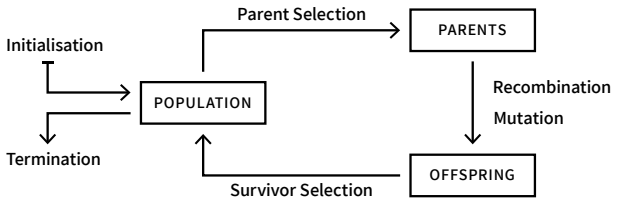

Figure 1. Evolutionary algorithms flow-chart.

work (CNN) that without any prior knowledge is effective on the classification of the CIFAR-10 dataset, with an average accuracy of 94.27%;

- The demonstration that ANNs evolved with DENSER generalise well. In concrete, an average accuracy of 78.75% on the CIFAR-100 dataset is obtained by a network whose topology was evolved for the CIFAR-10 dataset. To the best of our knowledge, this is the best result reported on the CIFAR-100 dataset by methods that automatically design CNNs.

The best trained models have been released at `http://github.com/fillassuncao/denser-models`.

The remainder of the paper is organised as follows. In Section 2 we introduce the concepts needed for understanding DENSER. Next, in Section 3, we describe our novel approach, DENSER, which is followed by the conducted set of experiments (reported in Section 4). To end, in Section 5, conclusions are drawn and future work is addressed.

2. Related Work

DENSER is a NeuroEvolution (NE) approach, and thus is based on the use of Evolutionary Algorithms (EAs) to automatically optimise ANNs. Next, we detail the principles behind EAs and review works that are closely related to ours.

2.1. Evolutionary Algorithms

EAs are stochastic search procedures inspired by the principles of natural selection and genetics, that have been successfully applied in optimisation, design and learning problems (Eiben & Smith, 2015). Historically, there are several variants of EAs, but they share the same common underlying idea: the simulation of evolution of a population of artificial individuals by natural selection (proposed by Darwin) via the application of selection, variation operators (typically crossover and mutation), and reproduction. These components are guided by a fitness function that evaluates each individual, measuring the quality of the solution it represents. The application of these components is repeated for several iterations, and over time it is expected that the overall quality of the individuals in the population improves. The process stops when a predetermined termination criterion is met (e.g., when a maximum number of iterations is achieved). Each artificial individual in EAs encodes a single candidate solution to the problem being considered.

The general flow-chart of a simple EA is shown in Figure 1. The main components are:

Representation – defines how the solutions the solutions to the problem under consideration should be encoded. The genetic material used to represent the solutions is known as genotype; the actual expression of the genotype is the phenotype.

Evaluation – estimates how good a solution is in solving the considered problem. Usually it is a mathematical expression and enables the comparison between problem solutions.

Parent Selection – selects, probabilistically, the population individuals (called parents) to participate in the breeding of a new population.

Variation Operators – create new individuals (offspring) using the parents. These operators are used in a stochastic manner, and are usually divided in two types: crossover, and mutation. Crossover creates variation in the population by taking two, or more individuals, as input, and rearranges their information to create new solutions. Mutation takes one individual as input and slightly modifies it.

Survivor Selection – determines the solutions that proceed to the next iteration of the EA. The number of individuals in an EA is typically kept fixed.

2.2. NeuroEvolution

NeuroEvolution (NE) (Floreano et al., 2008) is a sub-field of Machine Learning (ML) and Evolutionary Computation (EC) that applies evolutionary methods to the optimisation of ANNs. NE approaches are commonly grouped according to the aspects of the ANNs that they optimise: (i) learning (Radi & Poli, 2003; Gomez et al., 2008; Morse & Stanley, 2016); (ii) topology (Harp et al., 1990; Soltanian et al., 2013; Rocha et al., 2007); or (iii) both topology and learning (Whitley et al., 1990; Stanley & Miikkulainen, 2002; Turner & Miller, 2013).

The vast majority of NE works target the evolution of small networks for very specific tasks. With DENSER our goal is to evolve large scale networks that can deal with vast amounts of data and challenging tasks. As an example, consider the VGG network (Simonyan & Zisserman, 2014): a 16 to 19 deep CNN that is often used for image recognition tasks. The number of neurons and connections involved

in a deep architectures usually turns connection (Kitano, 1990; Leung et al., 2003; Fernando et al., 2017) or node-based (Moriarty & Miikkulainen, 1997; Stanley & Miikkulainen, 2002; Assunção et al., 2017b) evolutionary methods impractical for the discovering high performing networks, due to the large search sapce that needs to be traversed. Therefore, for evolving deep networks practitioners often resort to layer-based encodings (Jung & Reggia, 2006; Suganuma et al., 2017; Miikkulainen et al., 2017). For similar reasons, it is unfeasible to directly evolve the weights of the networks, which easily reach the range of thousands, or even millions of parameters; when the training of the networks is optimised using EC usually only the hyper-parameters are tuned and the networks trained using gradient-descent algorithms (Miikkulainen et al., 2017; Suganuma et al., 2017).

Loshchilov and Hutter (Loshchilov & Hutter, 2016) developed an approach based on Evolutionary Strategies (Hansen & Ostermeier, 2001) to optimise the hyper-parameters of deep networks; the tuned parameters are concerned with the topology (e.g., number of filters in the convolution layers, and number of neurons in fully-connected layers) and learning (e.g., batch size, and learning rates). This approach requires the *a-priori* definition of the network structure that may be suitable for solving the problem, and consequently there is no optimisation of the sequence of layers and connections between them.

The idea of optimising hyper-parameters for deep networks is further extended in Coevolution DeepNEAT (CoDeepNEAT) (Miikkulainen et al., 2017), where the structure of the network is searched combining the ideas behind Symbiotic, Adaptive Neuro-Evolution (SANE) (Moriarty & Miikkulainen, 1997) and NeuroEvolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen, 2002). Two populations are evolved in simultaneous: one of modules and another one of blueprints, which specify the modules that should be used. Learning and data augmentation parameters are also optimised.

A similar approach is proposed in CGP-NN (Suganuma et al., 2017), where Cartesian Genetic Programming (CGP) (Miller & Thomson, 2000) is used in the evolution of the architecture of CNNs. However, instead of automatically searching the modules, they are specified by the user, and just their combination and parameterisation is evolved.

In this work we want to make the automatic generation of deep networks as easy and transparent as possible. That is the reason why we adopt a grammar-based approach. Grammars allow us to specify different network types, such as AutoEncoders and CNNs) without the need to change any implementation details. Further, grammar-based methods make it easy to incorporate knowledge, allowing the specification of modules and/or parameters that we may know or suspect that work well on certain problem domains.

Table 1. Performance of different models on the classification of the CIFAR-10 and CIFAR-100 datasets. Evolutionary approaches are marked with an *.

Approach	Dataset	Accuracy
(Loshchilov & Hutter, 2016)*	CIFAR-10	$\sim 90.7\%$
(Miikkulainen et al., 2017)*	CIFAR-10	92.7%
(Snoek et al., 2015)	CIFAR-10	93.63%
(Suganuma et al., 2017)*	CIFAR-10	94.02%
(Real et al., 2017)*	CIFAR-10	94.60%
(Graham, 2014)	CIFAR-10	96.53%
(Snoek et al., 2015)	CIFAR-100	72.60%
(Graham, 2014)	CIFAR-100	73.61%
(Real et al., 2017)*	CIFAR-100	77.00%

There are several approaches concerned with the evolution of ANNs using Grammatical Evolution (GE) (O'Neill & Ryan, 2003). The vast majority of them focus on the tuning of a single hidden-layer, i.e., on the number of neurons and their connections from the input to the hidden-nodes and from the hidden-nodes to the outputs (Soltanian et al., 2013; Ahmadizar et al., 2015; Assunção et al., 2017a). In (Assunção et al., 2017b) a grammar-based method that is able to evolve networks with more than one hidden-layer is described. Despite theoretically suit to generate deep networks, the involved amount of neurons and connections make the domain space too large to be searched within an acceptable time.

Table 1 reports on the performance of different models for the classification of the CIFAR-10 and CIFAR-100 dataset. Those models discovered by evolutionary approaches are marked with an *. We only report results for this two datasets, since they are the ones where experiments will be conducted using DENSER.

3. Deep Evolutionary Network Structured Representation

Deep Evolutionary Network Structured Representation DENSER is a novel representation that combines the basic principles of two EAs: Genetic Algorithms (GAs) (Mitchell, 1998) and Dynamic Structured Grammatical Evolution (DSGE) (Assunção et al., 2017b). DSGE is a variant of GE (O'Neill & Ryan, 2003; Lourenço et al., 2016), and thus a form of Genetic Programming (GP) (Koza, 1992). Whilst in the vanilla version of GP the solutions are encoded via a syntax-tree, in GE approaches the candidate solutions are encoded using variable length integer arrays, which represent derivations of a user-defined Context-Free-Grammar (CFG). Formally, a CFG 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, $S \in N$ is the starting symbol, and P is the set of production rules of the form $A ::= \alpha$, with $A \in N$ and $\alpha \in (N \cup T)^*$. N and T are disjoint. Each

```
<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]
```

Figure 2. Example grammar for the encoding of CNNs.

grammar G defines a language $L(G)$ composed by all sequences of terminal symbols that can be derived from the starting symbol: $L(G) = \{w : S \overset{*}{\Rightarrow} w, w \in T^*\}$. In the upcoming sub-sections we further detail each component of DENSER.

3.1. Representation

Each solution encodes an ANN by means of an ordered sequence of feedforward layers and their respective parameters; the learning and any other hyper-parameters can be encoded with each individual too. The representation of the candidate solution is made at two different levels:

GA Level – encodes the macro structure of the networks and is responsible for representing the sequence of layers that later serves as an indicator to the grammatical starting symbol. It requires the definition of the allowed structure of the networks, i.e., the valid sequence of layers. For example, for evolving CNNs the following GA structure may be specified: [(features, 1, 10), (classification, 1, 2), (softmax, 1, 1), (learning, 1, 1)], where each tuple indicates the valid starting symbols, and the minimum and maximum number of times they can be used. Using the grammar of Figure 2, this GA structure can evolve networks with up to 10 convolution or pooling layers, followed by up to 2 fully-connected layers, and the classification layer softmax, that usually has a number of outputs that matches the number of problem classes; the learning tuple is responsible for codifying the parameters that should be used to train the network.

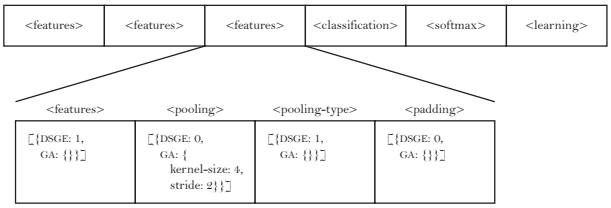

Figure 3. Example of the genotype of a candidate solution that encodes a CNN.

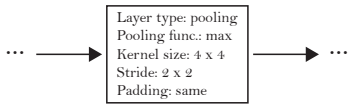

Figure 4. Phenotype corresponding to the only layer specified in Figure 3.

DSGE Level – encodes the parameters associated to a layer. The parameters and their allowed values or ranges are codified in the grammar that must be defined by the user. Looking at the grammar of Figure 2, for the pooling layers we tune the kernel size, the stride, and the type of padding. The same exercise can be made to the remaining layers defined in the grammar. The parameters can have closed values (e.g., the padding that can be only valid or same), or can assume a value in an integer or real interval.

The novel combination of a GA with DSGE enables a two-fold gain: (i) the GA level encapsulates the genetic material of each layer, making it easier to apply the variation operators (described next); and (ii) the DSGE makes the approach easily generalisable, as it is only needed to change the grammar to enable the evolution of different types of networks, or of networks to solve different tasks; it also facilitates the incorporation of domain specific knowledge.

An example of the genotype is shown in Figure 3. This example is based on the grammar of Figure 2 and on the above detailed GA structure. Figure 4 depicts the phenotype corresponding to the layer which has the DSGE genotype detailed in Figure 3.

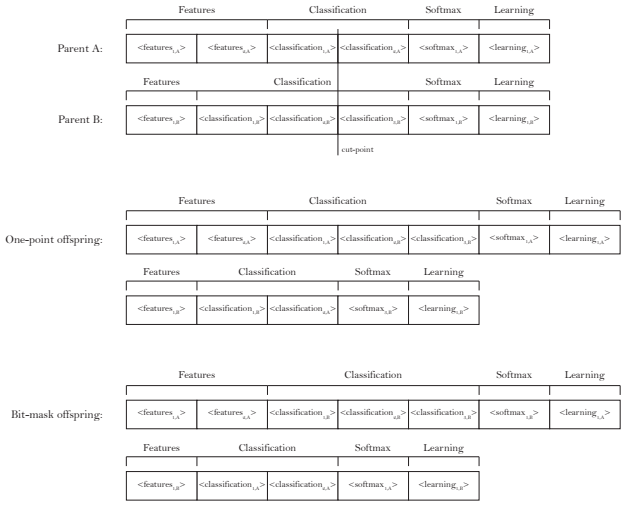

Figure 5. Example of the introduced crossover operators. The example focuses on the GA level of the genotype. For the bit-mask crossover the mask is 1001, which is associated to the features, classification, softmax and learning modules, respectively.

3.2. Variation Operators

To promote the evolution of the solutions we propose variation operators designed to operate on the two levels of the genotype.

Crossover

We designed two crossover operators, both of them based on the premise that the genetic material is encapsulated, which facilitates the exchange betwenn individuals. In the context of this work a 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 structure index.

To exchange layers within the same module we use a one-point crossover. Imagining that we are evolving bitstrings, if the parents are $111|000$ and $101|010$, and $|$ represents the cut-point the offspring results from changing the genetic material delimited by the cut-point, i.e., 111010 and 101000. The same module in different individuals may vary in size; the cut-point is generated at random taking into account the smallest module.

To exchange modules between two parents we use a bit-mask crossover. In the bitmask crossover, as the name suggests first we need to create a bitmask of the size of the number of number of codons (i.e., modules) that are to be exchanged (in the case of the above GA structure, 4). Then the offspring is created by copying a codon from the first parent if the bit is 1, and if the bit is 0 the codon is copied from the second parent.

An example of the application of the crossover operators is depicted in Figure 5.

Mutation

We develop two sets of mutation operators that act upon the GA and DSGE levels, respectively. The mutations on the GA level are three:

Add layer – generates a new layer randomly, subjected to the constraints of the module where it will be placed.

Replicate layer – selects a module and copies an existing layer to another position of the module; the copy is made by reference, meaning that any change in the parameters of the layer is propagated to the copies.

Remove layer – deletes a random layer from a module, without violating the minimum number of layers.

The mutations on the DSGE level are two:

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

Integer/float mutation – an integer/float block is replaced by a new one. For integers we generate new integers at random; for floats a Gaussian perturbation is used.

3.3. Evaluation

In DENSER, as in the majority of layer-based NE approaches, we only allow the evolution of the learning hyperparameters. Therefore, to evaluate the candidate solutions we must train them on the task that we are trying to solve. In this work we are going to perform object recognition tasks, using the CIFAR-10 dataset (Krizhevsky & Hinton, 2009). The quality of the solutions is measured base on the accuracy metric. The training and evaluation is performed by mapping the solutions generated by DENSER, i.e., strings that result from the grammatical derivation of the defined grammar, into a Keras model running on top of TensorFlow, and then executed on a GPU. Each network is trained during 10 epochs, and in the end we return the best accuracy on the validation set.

While in traditional ML approaches we often only need two dataset partitions (train and test), in NE we need three:

Train – used to train the network using the defined or evolved learning parameters;

Validation – used to evaluate the performance of the network during evolution, i.e., the accuracy on the validation set is used as the fitness metric;

Table 2. 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
Training Parameter	**Value**
Number of epochs	10
Loss	Categorical Cross-entropy
Batch size	125
Learning rate	0.01
Momentum	0.9
Data Augmentation Parameter	**Value**
Padding	4
Random crop	4
Horizontal flipping	50%

Test – kept aside from the evolutionary process, and used to evaluate the performance of the best models on unseen data, so we can understand the generalisation ability.

If we define no test set and only use two partitions it is impossible to test the evolved models on data that is not presented to the model during evolution. Thus the reported results would be biased. Cross-validation is not applied due to the associated computational cost. Moreover, we followed the same data augmentation method reported by (Suganuma et al., 2017), which includes: padding, horizontal flips, and random crops.

4. Experiments

To analyse the performance of DENSER we perform experiments on the generation of CNNs for the classification of the CIFAR-10 dataset. Then, to assess the generalisation and scalability ability of the networks that are evolved using DENSER we take the best model found for the classification of the CIFAR-10 dataset and re-train it on the CIFAR-100.

4.1. Dataset

The CIFAR-10 and CIFAR-100 datasets (Krizhevsky & Hinton, 2009) are composed by 60000 32×32 colour images. The aim is to maximise the accuracy of the object recognition task, i.e., the number of images that are assigned the right class. The datasets have 10 and 100 disjoint classes, respectively.

4.2. Experimental Setup

The experimental parameters used are detailed in Table 2. We use the grammar of Figure 2 and the following GA structure: [(features, 1, 30), (classification, 1, 10), (softmax, 1, 1)]. This way our search space encompasses CNNs that have up to 40 hidden-layers: at most 30 convolution or pooling layers followed by up to 10 fully-connected layers.

The training of a CNNs is a computationally expensive task. For that reason, during the evolutionary process we only perform 10 training epochs for each network. In the end of each evolutionary run, we take the best network and perform longer trains, with 400 epochs and the same learning rate policy. For these extended trains, the train and validation sets are merged used in the evolutionary runs, so that more data is available for learning.

4.3. Evolution of CNNs for the CIFAR-10 Dataset

The fitness evolution of the best networks across the generations is depicted in Figure 6. An inspection of the results shows that the performance of the networks is steadily increasing, and evolution converges around the 80th generation. It is possible to see that a change in behaviour occurs around the 60th generation: before the increase in fitness is followed by a decrease in the number of hidden-layers; after the increase in fitness is accompanied by an increase in the number of hidden-layers. To support this analysis we compute the Pearson correlation between the fitness values of the best individuals, and the average number of layers, per generation. Until the 60th generation the Pearson correlation reports a coefficient of -0.7166 (moderate negative correlation); after the 60th generation the correlation coefficient is 0.9204 (strong positive correlation).

The change in behaviour around the 60th generation is easily explainable. At first, the candidate solutions are randomly generated and have approximately 22.44 hidden-layers (population average). These initial solutions have their parameters generated at random, and thus, it is highly unlikely that a stochastic parameterisation of 22 layers will have any meaningful results. As time passes, the networks decrease in complexity and their parameters are tuned, until a point where to increase the performance of the networks more layers are necessary (60th generation).

Once the evolutionary process is over, the best network of each run is re-trained 5 times during 400 epochs. The best network of each run is selected according to the accuracy values on the validation set. All accuracy values reported from this point onwards are averages of 5 independent trains; this is done because the initial weights are different. This training conditions lead to an average classification accuracy on the test set of 88.41% (error of 11.59%).

The comparison of this result with the ones reported by

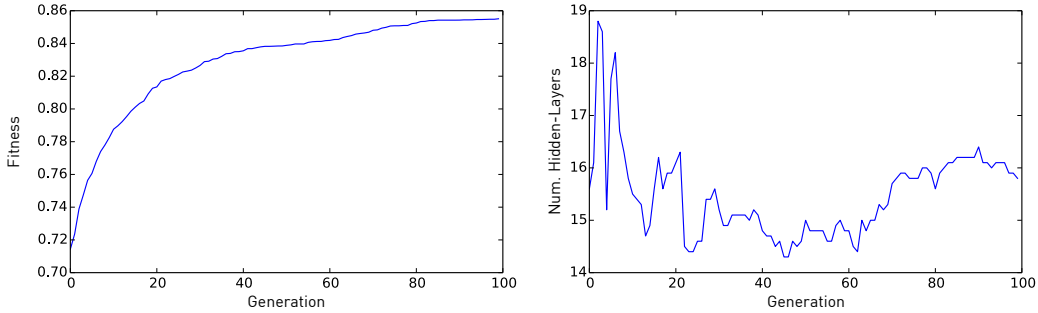

Figure 6. Evolution of the fitness (left) and number of hidden-layers (right) of the best individuals across generations. Results are averages of 10 independent runs.

other approaches (check Table 1) seems to indicate that DENSER performs worse. However, different approaches use different training policies, data augmentation, or prediction techniques. For example, (Snoek et al., 2015) for each instance of the test set generate 100 augmented images, and the prediction of the image class is based on the class that has the maximum average confidence on the 100 augmented images. By doing this the average accuracy on the test set increases to 89.93% (error of 10.07%).

To check if it is still possible to enhance the performance of the best networks we test a different learning rate policy. In (Suganuma et al., 2017) the authors use a varying learning rate policy: 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; Nesterov momentum is used. The previous changes lead to an average test accuracy of 92.51% (error of 7.49%), not using data augmentation on the test set. Applying data augmentation on the test, the performance further increases the average test accuracy to 93.29% (error of 6.71%).

Recall that all the previous results are averages of the 10 best networks, one from every of the performed evolutionary runs. Each network is trained 5 times. The average accuracy on the test set of the network that has the highest accuracy on 5 trains is 93.37% (not using data augmentation on the test set), or 94.13% (using data augmentation of the test set). These results are with the learning rate policy of (Suganuma et al., 2017). If we focus on only the single train that reports the highest accuracy on the test set further increases to 94.27% (error of 5.73%).

Based on all the previous results it is possible to conclude that DENSER is the evolutionary approach that reports the lowest error on the CIFAR-10 dataset. The number of trainable parameters of the best network is around 10.81×10^6, which is much higher in our methodology[1] because we enable the generated CNNs to have fully-connected layers after the convolution and/or pooling layers. Another key aspect is that these results are obtained with limited computational power (the networks are evaluated on just 10 epochs), and no prior-knowledge regarding possible effective network structures is used in the definition of the used grammar.

Focusing on the structure of the best networks, the most puzzling characteristic is the importance and number of the fully-connected layers. On average, the best networks have 2.2 fully-connected layers that are placed before the softmax layer. We tested removing some of the fully-connected layers and preliminary results show that the performance of the networks degenerates. This outcome of the evolution is remarkable, since the majority of hand-designed networks only have one dense layer, i.e., the classification layer. Figure 7 depicts the network that reports the best accuracy on the test set.

4.4. Generalisation to the CIFAR-100 Dataset

To test the generalisation and scalability of the evolved networks we take the best network generated by DENSER on the CIFAR-10 dataset and apply it to the classification of the CIFAR-100 dataset. In order for the network to work on the CIFAR-100 dataset we only change the softmax layer to have 100 output neurons, instead of 10.

The best results on the CIFAR-100 are obtained using the same learning rate policy as in (Suganuma et al., 2017), i.e., a varying learning rate that starts at 0.01, and that on the 5th, 250th and 375th epochs changes to 0.1, 0.01, and 0.001, respectively. With this learning policy, the average test accuracy of 5 independent trains of the best network is of 73.32%, with no data augmentation on the test set. If we apply the same method as before, i.e., augment each test instance 100 times, the average test accuracy increases to 74.94%.

The training of each networks is stochastic; the initial conditions are different and they are trained using different instances of the dataset, because of the data augmentation process. Thus and in order to further improve the results, we

[1]The best network of CGP-CNN (Suganuma et al., 2017) has 1.68×10^6 trainable parameters.

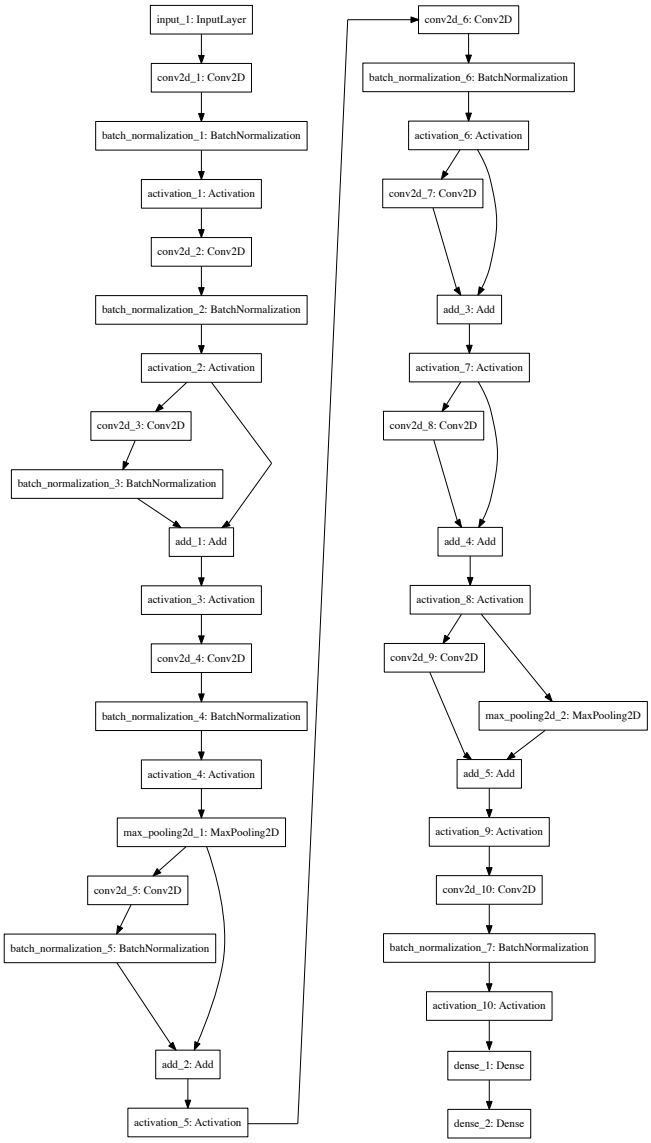

Figure 7. Topology of the CNN that reports the best results.

investigate if the approach proposed in (Graham, 2014) to test the performance of the fractional max-pooling increases the performance reported by our network. In brief words, instead of a single network, an ensemble is used, where each network that is part of the ensemble is the result of an independent training of the network. Using this methods, an ensemble of the same network trained 5 times reports a test accuracy of 77.51%, an ensemble of 10 trains a test accuracy of 77.89%, and an ensemble of 12 trains a test accuracy of 78.14%. These results outperform those reported in the literature for the evolution of CNNs with a fairly standard data augmentation methodology.

Moreover, and instead of forming ensembles with the same network structure we also tested the performance of building an ensemble using by the two best network topologies found by DENSER, similarly to what is done in (Real et al., 2017). Following this methodology, we obtain were able to increase the accuracy to 78.75%.

5. Conclusions and Future Work

The large amounts of data and the complex tasks that researchers are trying to solve often make the use of ANNs with deep architectures necessary. The challenge when developing these type of networks relies on the difficulty to find adequate structures and their parameterisation. To tackle this issue, methods that address the automatic generation of ANNs by means of evolutionary methods have been proposed. However, the majority of them are not capable of dealing with the evolution of large scale deep structures, or are too constraint for a specific type of network architecture.

In the current article we propose DENSER: a layer-based NE approach that combines a GA with DSGE. The representation of the candidate solutions in DENSER, i.e., deep networks, is made at two different levels: the GA level, that encodes the ordered linear sequence of layers; and the DSGE level, which encodes the parameters of each single layer. We also design specific genetic operators that focus on the evolution of ANNs based on the two-level representation of the genotype. The way in which the solutions are encoded allows a two-fold gain: (i) the genetic material is encapsulated, which facilitates the application of the genetic operators; and (ii) the grammatical nature of the method that makes it easy to evolve solutions to different problems, or different network structures.

The results show the effectiveness of DENSER. We conduct experiments in the evolution of CNNs for the CIFAR-10 dataset, and the results show that DENSER is currently the evolutionary approach, with no prior-knowledge, that generates the highest performing networks, with an accuracy of up to 94.27%. Neverthless, the most striking result is the scalability capacity of the networks evolved by DENSER. We trained the best network evolved by DENSER for the CIFAR-10 dataset and used it for the classification of the CIFAR-100 dataset. The network not only is capable of classifying the CIFAR-100 datasets, but does so with an impressive accuracy of 78.75%, which is the best result reported by NE approaches on the CIFAR-100 dataset.

For future work we intend to study how to improve the optimisation of the learning parameters. The evaluation of the networks is being conducted on just 10 epochs, and thus it is hard to evolve effective learning rate schedules. Moreover we plan on studying the impact of evolving modular structures, i.e., instead of replicating just single layers allow the repetition of a set of layers multiple times in the network.

Acknowledgements

This work is partially funded by: Fundação para a Ciência e Tecnologia (FCT), Portugal, under the grant SFRH/BD/114865/2016. We would also like to thank NVIDIA for providing us Titan X GPUs.

References

Ahmadizar, Fardin, Soltanian, Khabat, AkhlaghianTab, Fardin, and Tsoulos, Ioannis. Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. *Engineering Applications of Artificial Intelligence*, 39:1–13, 2015.

Assunção, Filipe, Lourenço, Nuno, Machado, Penousal, and Ribeiro, Bernardete. Evolving the topology of large scale deep neural networks. In *EuroGP*, volume 1. Springer, 2017a.

Assunção, Filipe, Lourenço, Nuno, Machado, Penousal, and Ribeiro, Bernardete. Towards the evolution of multi-layered neural networks: A dynamic structured grammatical evolution approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pp. 393–400, New York, NY, USA, 2017b. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071286. URL http://doi.acm.org/10.1145/3071178.3071286.

Eiben, Agoston and Smith, James. *Introduction to evolutionary computing*. Natural computing series. Springer, Berlin, Heidelberg, Paris, 2015. ISBN 3-540-40184-9.

Fernando, Chrisantha, Banarse, Dylan, Blundell, Charles, Zwols, Yori, Ha, David, Rusu, Andrei A., Pritzel, Alexander, and Wierstra, Daan. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Floreano, Dario, Dürr, Peter, and Mattiussi, Claudio. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

Frean, Marcus. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990.

Gomez, Faustino, Schmidhuber, Jürgen, and Miikkulainen, Risto. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008.

Graham, Benjamin. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

Hansen, Nikolaus and Ostermeier, Andreas. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Harp, Steven A, Samad, Tariq, and Guha, Aloke. Designing application-specific neural networks using the genetic algorithm. In *Advances in neural information processing systems*, pp. 447–454, 1990.

Jung, Jae-Yoon and Reggia, James A. Evolutionary design of neural network architectures using a descriptive encoding language. *IEEE transactions on evolutionary computation*, 10(6):676–688, 2006.

Kitano, Hiroaki. Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4(4):461–476, 1990.

Koza, John R. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images, 2009.

Leung, Frank Hung-Fat, Lam, Hak-Keung, Ling, Sai-Ho, and Tam, Peter Kwong-Shun. 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.

Loshchilov, Ilya and Hutter, Frank. CMA-ES for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

Lourenço, Nuno, Pereira, Francisco B, and Costa, Ernesto. Unveiling the properties of structured grammatical evolution. *Genetic Programming and Evolvable Machines*, 17 (3):251–289, 2016.

Miikkulainen, Risto, Liang, Jason, Meyerson, Elliot, Rawal, Aditya, Fink, Dan, Francon, Olivier, Raju, Bala, Navruzyan, Arshak, Duffy, Nigel, and Hodjat, Babak. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

Miller, Julian F and Thomson, Peter. Cartesian genetic programming. In *European Conference on Genetic Programming*, pp. 121–132. Springer, 2000.

Mitchell, Melanie. *An introduction to genetic algorithms*. MIT press, 1998.

Molchanov, Pavlo, Tyree, Stephen, Karras, Tero, Aila, Timo, and Kautz, Jan. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 2016.

Moriarty, David E. and Miikkulainen, Risto. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399, 1997.

Morse, Gregory and Stanley, Kenneth O. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pp. 477–484. ACM, 2016.

O'Neill, Michael and Ryan, Conor. Grammatical evolution. In *Grammatical Evolution*, pp. 33–47. Springer, 2003.

Parekh, Rajesh, Yang, Jihoon, and Honavar, Vasant. Constructive neural-network learning algorithms for pattern classification. *IEEE Transactions on Neural Networks*, 11(2):436–451, 2000.

Radi, Amr and Poli, Riccardo. Discovering efficient learning rules for feedforward neural networks using genetic programming. In *Recent advances in intelligent paradigms and applications*, pp. 133–159. Springer, 2003.

Real, Esteban, Moore, Sherry, Selle, Andrew, Saxena, Saurabh, Suematsu, Yutaka Leon, Le, Quoc, and Kurakin, Alex. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

Reed, Russell. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

Rocha, Miguel, Cortez, Paulo, and Neves, José. Evolution of neural networks for classification and regression. *Neurocomputing*, 70(16):2809–2816, 2007.

Sietsma, Jocelyn and Dow, Robert JF. Creating artificial neural networks that generalize. *Neural networks*, 4(1): 67–79, 1991.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Mostofa, Prabhat, Mr, and Adams, Ryan. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pp. 2171–2180, 2015.

Soltanian, Khabat, Tab, Fardin Akhlaghian, Zar, Fardin Ahmadi, and Tsoulos, Ioannis. Artificial neural networks generation using grammatical evolution. In *Electrical Engineering (ICEE), 2013 21st Iranian Conference on*, pp. 1–5. IEEE, 2013.

Stanley, Kenneth O. and Miikkulainen, Risto. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Suganuma, Masanori, Shirakawa, Shinichi, and Nagao, Tomoharu. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pp. 497–504, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071229. URL http://doi.acm.org/10.1145/3071178.3071229.

Turner, Andrew James and Miller, Julian Francis. Cartesian genetic programming encoded artificial neural networks: a comparison using three benchmarks. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 1005–1012. ACM, 2013.

Whitley, Darrell, Starkweather, Timothy, and Bogart, Christopher. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14(3):347–361, 1990.