Fast-DENSER++: Evolving Fully-Trained Deep Artificial 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

This paper proposes a new extension to Deep Evolutionary Network Structured Evolution (DENSER), called Fast-DENSER++ (F-DENSER++). The vast majority of NeuroEvolution methods that optimise Deep Artificial Neural Networks (DANNs) only evaluate the candidate solutions for a fixed amount of epochs; this makes it difficult to effectively assess the learning strategy, and requires the best generated network to be further trained after evolution. F-DENSER++ enables the training time of the candidate solutions to grow continuously as necessary, i.e., in the initial generations the candidate solutions are trained for shorter times, and as generations proceed it is expected that longer training cycles enable better performances. Consequently, the models discovered by F-DENSER++ are fully-trained DANNs, and are ready for deployment after evolution, without the need for further training. The results demonstrate the ability of F-DENSER++ to effectively generate fully-trained DANNs; by the end of evolution, whilst the average performance of the models generated by F-DENSER++ is of 88.73%, the performance of the models generated by the previous version of DENSER (Fast-DENSER) is 86.91% (statistically significant), which increases to 87.76% when allowed to train for longer.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Supervised learning by classification*; Object recognition.

KEYWORDS

Convolutional Neural Networks, Deep Evolutionary Network Representation, NeuroEvolution

1 INTRODUCTION

Automated Machine Learning (AutoML) seeks to model with little or no human-intervention the application of Machine Learning (ML) techniques to well defined problems, avoiding the user to manually perform the data pre-processing, the design and extraction of features, and/or the selection and parameterisation of the most suit ML model. The current paper focuses on a branch of AutoML: NeuroEvolution (NE) [3]. NE automatically searches for Artificial Neural Networks (ANNs), enabling the optimisation of their structure (i.e.., number of neurons, layers, and/or connectivity), and/or learning strategy (i.e., learning algorithm and its parameters: e.g., learning rate, momentum); in NE, Evolutionary Computation (EC) is used to automate the search for ANNs.

Considering that NE is based on EC, a population of individuals is continuously evolved throughout generations; each single individual encodes an ANN. One of the main drawbacks relies on the time that is required to evaluate the population, which is even higher if we consider deep networks. To overcome this issue the vast majority of NE methods constraint the evaluation of the networks to a fixed (low) number of epochs, or grant the networks a limited amount of Graphic Processing Unit (GPU) training time. However, these evaluation strategies cannot assure that the candidate solutions are being trained for the required time, and make it difficult to assess the quality of the evolved learning strategy, i.e., that the generated learning strategy works beyond the limited number of epochs / GPU training time.

To overcome the previous limitation, in this paper we propose a new version of Deep Evolutionary Network Structure Representation (DENSER), called Fast-DENSER++ that enables the evaluation time to grow continuously as the complexity of the networks increases throughout the generations. Fast-DENSER++ (F-DENSER++) is an extension to Fast-DENSER (F-DENSER): a previous version of DENSER that generates networks with the same performance of DENSER, but 20x faster. The results demonstrate that F-DENSER++ is statistically superior to and F-DENSER in evolutionary performance; further, when the F-DENSER networks are trained for longer they achieve, on average, lower performances than those reported by F-DENSER++. That is, F-DENSER++ is able

```
<fully-connected> ::= layer:fc <activation>                                  (1)
                      [num-units,int,1,128,2048 <bias>                       (2)
<dropout> ::=layer:dropput [rate,float,1,0,0.7]                              (3)
<activation> ::= act:linear | act:relu | act:sigmoid                         (4)
<bias> ::= bias:True | bias:False                                            (5)
<softmax> ::= layer:fc act:softmax num-units:10 bias:True                    (6)
<learning> ::= <bp> [batch_size,int,1,50,500]                                (7)
               | <rmsprop> [batch_size,int,1,50,500]                         (8)
               | <adam> [batch_size,int,1,50,500]                            (9)
<bp> ::= learning:gradient-descent [lr,float,1,0.0001,0.1]                   (10)
         [momentum,float,1,0.68,0.99]                                        (11)
         [decay,float,1,0.000001,0.001] <nesterov>                           (12)
<nesterov> ::= nesterov:True | nesterov:False                                (13)
<adam> ::= learning:adam [lr,float,1,0.0001,0.1]                             (14)
           [beta1,float,1,0.5,1] [beta2,float,1,0.5,1]                       (15)
           [decay,float,1,0.000001,0.001]                                    (16)
<rmsprop> ::= learning:rmsprop [lr,float,1,0.0001,0.1]                       (17)
              [rho,float,1,0.5,1] [decay,float,1,0.000001,0.001]             (18)
```

Figure 1: Example of a grammar for encoding fully-connected networks.

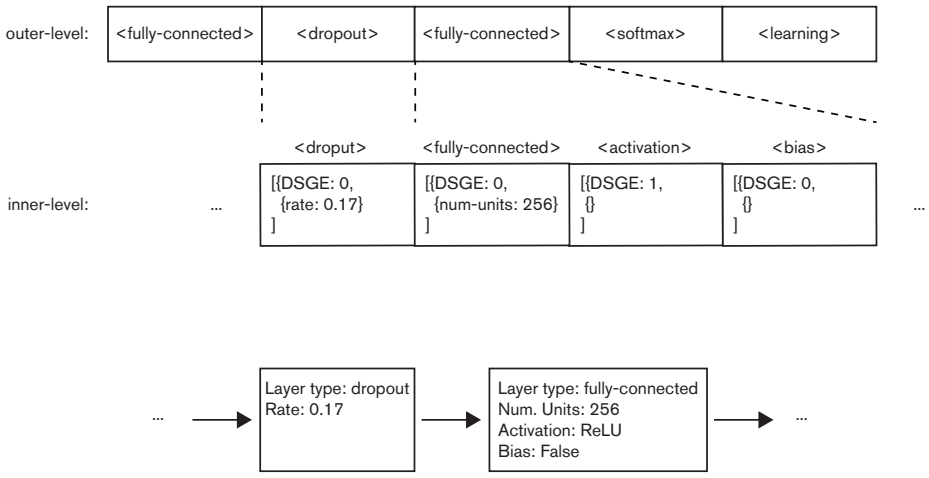

Figure 2: DENSER genotype (top), and respective phenotype (bottom). The example is based on the outer-level structure [((fully-connected, dropout), 1, 10), (softmax, 1, 1), (learning, 1, 1)], and on the grammar of Figure 1.

to effectively generate models that are ready for deployment after the end of evolution, without the need for additional training.

The remainder of the paper is organised as follows. Section 2 briefly surveys NeuroEvolution works 2. Section 3 presents DENSER, and its extension F-DENSER. Section 4 details F-DENSER++, and the experimental setup and results. Section 5 draws conclusions and addresses future work.

2 NEUROEVOLUTION

NeuroEvolution (NE) approaches usually focus on the evolution of the learning strategy [4, 13, 16, 20] or the topology [5, 10]. On the optimisation of the learning strategy NE has been able to match and even surpass the results attained by standard learning algorithms [12]; on the other hand, the automatic optimisation of the topology by NE is faster and finds better solutions than using grid or random search [8]. Nonetheless, when designing a network from the scratch it is hard to separate the learning from the topology, as both are correlated in what regards the search for the most effective model to solve a specific task. Examples of NE approaches that have successfully addressed the simultaneous optimisation of the learning and topology are [11, 17, 19].

Although the methods are commonly grouped as above, according to the target of evolution, more recent efforts have been put towards the development of methods that are capable of dealing with Deep Artificial Neural Networks (DANNs), and thus we feel that it is more intuitive to divide them into small-scale [17, 20] and large-scale [1, 9, 15, 16, 18] NE. The current paper focuses on the latter; more specifically we will extend F-DENSER [2] to enable the generation of models that can be used right-off evolution, without further training. F-DENSER is a general-purpose grammar-based NE approach that can be easily adapted to deal with different problems and/or network types; there is just the need to change the grammar that is feed to the system.

The problem of most of the methods that target the evolution of DANNs is that, even aided by Graphic Processing Units (GPUs) they tend to take a lot of time to find effective models. For example, CoDeepNEAT [9] train on 100 GPUs, and Real et al. use 450 GPUs for 7 days to perform each run [14]. F-DENSER takes approximately 55 hours (2.3 days) with a single GPU to perform each run, and that is the reason why we have selected F-DENSER for the current paper. There are methods that are computationally cheaper, e.g., Lorenzo and Nalepa [7] take about 120 minutes to obtain results; however, the speedup is obtained at the cost of the model performance.

3 DEEP EVOLUTIONARY NETWORK STRUCTURED REPRESENTATION

Deep Evolutionary Network Structured Representation (DENSER) [1] is a grammar-based general purpose NE method: it enables the automatic generation of the network topology (sequence, type, connectivity, and parameterisation of the network layers), and learning strategy (learning algorithm and parameterisation). To make this possible DENSER has a two-level representation: (i) the outer-level encodes an ordered sequence of evolutionary units[1]; and (ii) the inner-level encodes the parameters of each evolutionary unit. In simple words, each evolutionary unit points to a grammar start symbol, and the grammar itself has every single parameter, and the allowed values. The grammatical nature of DENSER makes the adaption to different network structures and problems easy and transparent, as the user only needs to change the grammar production rules, which are in a text human-readable format. In addition to the grammar, the user needs to define the outer-level structure, which sets the allowed network structure using the following format: [(production-rules, min_evo_units, max_evo_units), ...]. An example of an outer-level structure for encoding fully-connect networks is [((fully-connected, dropout), 1, 10), (softmax, 1, 1), (learning, 1, 1)], which defines fully-connected networks with between 2 and 11 layers, and a learning block.

[1]The evolutionary units can encode layers, learning strategies, or even data preprocessing and/or augmentation strategies.

```
parent ← select_fittest(population)
if parent.train_time > DEFAULT_TIME then
    tmp_parent ← select_fittest(population-parent)
    retrain(tmp_parent, parent.train_time)
    if tmp_parent.fitness > parent.fitness then
        return tmp_parent
    else
        return parent
    end
else
    return parent
end
```

Algorithm 1: Parent selection algorithm.

Evolution proceeds by a combination of a Genetic Algorithm (GA) with Dynamic Structured Grammatical Evolution (DSGE). The typical mutation operators of GAs are applied to the outer-level (add, remove, duplicate), and DSGE mutations are applied to the inner-level, i.e., change the expansion possibility, and parameters values. DSGE is chosen over standard Grammatical Evolution (GE) for its ability to deal with the locality and redundancy issues present in GE. To enhance locality the method introduces a one-to-one mapping between the genotype and the non-terminal symbols: there is a list of integers for each non-terminal symbol, and when decoding the genotype the expansion possibility is read from the corresponding list; because each non-terminal symbol has a list associated to it, there is no need for the modulus mathematical operation to select the expansion possibility, and thus redundancy is reduced.

An example of a grammar for encoding fully-connected networks is shown in Figure 1. The grammar encodes the parameters needed for each evolutionary unit, and are encoded according to the structure: [variable-name, variable-type, num_values, min_value, max_value]. The parameter type can be integer, or float; closed choice parameters are enabled using the grammatical expansion possibilities (e.g., line 5 of the grammar). Figure 2 represents an example of the genotype and phenotype of an individual using the above outer-level structure, and the grammar of Figure 1.

To speedup search, Fast-DENSER (F-DENSER) [2] was introduced: a representation with the same outer and inner-levels is used, and another level is created to encode the connectivity of each layer; this level is referred to as the connectivity-level. Therefore, F-DENSER can evolve not only feed-forward networks but also topologies where any given layer can receive multiple previous layers as input. The same mutation operators are applied to promote evolution, but additionally there are two new operators related to the connectivity-level, that add/remove inputs to layers. In F-DENSER the evolutionary engine is replaced by a (1+λ)-Evolutionary Strategy (ES). Therefore whilst in DENSER a typically large population of individuals needs to be evaluated, in F-DENSER there is just the need to evaluate (1+λ) individuals. The results have proved that, without sacrificing performance, F-DENSER with $\lambda = 4$ is 20x faster than the original DENSER implementation with a population size of 100 individuals. The previous results are achieved with the same evaluation method, i.e., each individual is trained for a fixed number of 10 epochs. In addition, with the rationale to grant all individuals the same computational resources evolution is conducted with the individuals being trained for a maximum GPU time of 10 minutes; this evaluation stop criteria leads to an improvement of the results.

Table 1: Experimental parameters.

Evolutionary Parameter	**Value**
Number of runs	10
Number of generations	150
Population size	5
Add layer rate	25%
Remove layer rate	25%
DSGE-level rate	15%
Dataset Parameter	**Value**
Train set	42500 instances
Validation set	7500 instances
Test set	10000 instances
Train Parameter	**Value**
Default train time	10 min.
Loss	Categorical Cross-entropy
Data Augmentation Parameter	**Value**
Padding	4
Random crop	4
Horizontal flipping	50%

4 FAST-DENSER++

Fast-DENSER++ is an extension to F-DENSER that enables the method to generate networks that are ready for deployment, i.e., the evolutionary result requires no further training to be used. To achieve this we introduce a new mutation operator that does not change any of the layer structure and/or learning parameters, and increases the train time of the individual. Whilst in F-DENSER the maximum train time is set the same for all individuals, in F-DENSER++ the maximum train time is set independently for each individual: in the initial population all individuals are trained for the same amount of time, and the mutation operator changes the maximum train time; any of the other mutation operators reset the evaluation time to the default value, so that the offspring solutions are not evaluated for longer than necessary.

The proposed mutation operator enables the train time to grow continuously as needed, i.e., during the initial generations the networks are simple and thus their train time is reduced, and as time passes more complex solutions require longer evaluations. On the other hand, the new operator makes it possible for individuals within the same population to have different evaluation times. This indirectly implies that the parent selection mechanism has to be changed, so that the comparison between the individuals in the population is fair. In case the fittest individual has been trained for the default train time, the selection is the same as before, i.e., the fittest individual seeds the next generation; otherwise, if the fittest individual is trained for longer than the default train time, the fittest individual of those that were trained for the default train time is re-trained, and the fittest among the two seeds the next generation. That is, the variations of the parent are initially evaluated

```
<features> ::= <convolution> | <convolution>                                  (1)
             | <pooling> | <pooling>                                          (2)
             | <dropout> | <batch-norm>                                       (3)
<convolution> ::= layer:conv [num-filters,int,1,32,256] [filter-shape,int,1,2,5] (4)
             [stride,int,1,1,3] <padding> <activation> <bias>                 (5)
<batch-norm> ::=layer:batch-norm                                              (6)
<pooling> ::= <pool-type> [kernel-size,int,1,2,5]                             (7)
             [stride,int,1,1,3] <padding>                                     (8)
<pool-type> ::= layer:pool-avg | layer:pool-max                               (9)
<padding> ::= padding:same | padding:valid                                    (10)
<classification> ::= <fully-connected> | <dropout>                            (11)
<fully-connected> ::= layer:fc <activation>                                   (12)
             [num-units,int,1,128,2048 <bias>                                 (13)
<dropout> ::=layer:dropput [rate,float,1,0,0.7]                               (14)
<activation> ::= act:linear | act:relu | act:sigmoid                          (15)
<bias> ::= bias:True | bias:False                                             (16)
<softmax> ::= layer:fc act:softmax num-units:10 bias:True                     (17)
<learning> ::= <bp> <early-stop> [batch_size,int,1,50,500]                    (18)
             | <rmsprop> <early-stop> [batch_size,int,1,50,500]               (19)
             | <adam> <early-stop> [batch_size,int,1,50,500]                  (20)
<bp> ::= learning:gradient-descent [lr,float,1,0.0001,0.1]                    (21)
             [momentum,float,1,0.68,0.99] [decay,float,1,0.000001,0.001]      (22)
             <nesterov>                                                       (23)
<nesterov> ::= nesterov:True | nesterov:False                                 (24)
<adam> ::= learning:adam [lr,float,1,0.0001,0.1] [beta1,float,1,0.5,1]        (25)
             [beta2,float,1,0.5,1] [decay,float,1,0.000001,0.001]             (26)
<rmsprop> ::= learning:rmsprop [lr,float,1,0.0001,0.1]                        (27)
             [rho,float,1,0.5,1] [decay,float,1,0.000001,0.001]               (28)
<early-stop> ::= [early_stop,int,1,5,20]                                      (29)
```

Figure 3: Grammar used by F-DENSER++, and F-DENSER for the evolution of CNNs for the CIFAR-10.

for the default time, and if in the population there is an individual evaluated for longer, the fittest individual is also granted the same time. The parent selection mechanism is clarified in Algorithm 1. Indirectly we are evolving solutions that have to train fast, but that given more time must improve performance.

To assess the ability of F-DENSER++ to generate ready to deploy DANNs we compare it to the F-DENSER implementation. Therefore we address the generation of Convolutional Neural Networks (CNNs) for the CIFAR-10 dataset. Section 4.1 details the experimental setup, and Section 4.2 the experimental results.

4.1 Experimental Setup

The experimental parameters are detailed in Table 1, and are divided into 4 sections: (i) evolutionary – (1+λ)-ES parameters; (ii) dataset – number of instances of each of the partitions of the dataset; the dataset is divided into three independent sets: (ii.i) the train set is used for training the individual during evolution; (ii.ii) the validation set is used to perform early stopping, and to assess the fitness of the candidate solution, and (ii.iii) the test is kept out of evolution and used only for assessing the performance of the individuals on unseen data; (iii) train – fixed training parameters; and (iv) parameters needed for data augmentation.

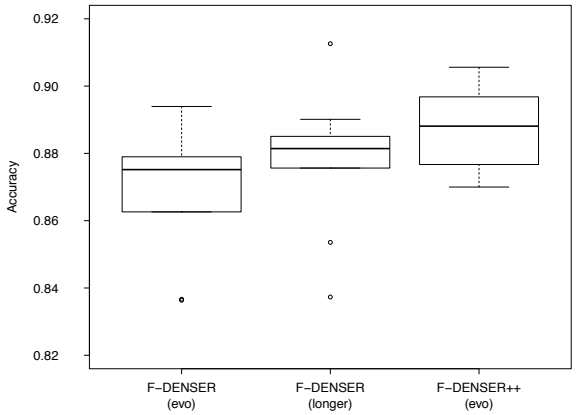

Figure 4: Box-plot of the test accuracies of F-DENSER (evo and longer), and F-DENSER++ (evo).

The reported parameters are the same for F-DENSER++ and F-DENSER, but additionally F-DENSER++ has another mutation rate, that defines the likelihood of an individual to be trained for longer, which is set to 20%.

The experiments contained in this section are conducted over the CIFAR-10 [6]: a dataset of 32×32 real-world images of objects. The CIFAR-10 is composed by 50000 train, and 10000 test instances. We search for CNNs for the CIFAR-10 using the grammar of Figure 3, that defines the search space for the topology, and learning strategy.

4.2 Experimental Results

The results of the evolution of CNNs for the CIFAR-10 with F-DENSER++, and F-DENSER are reported in Table 2. The evolutionary performance, i.e., fitness (validation), and on unseen data during evolution (test) are summarised; the values are the average of the 10 highest performing networks (according to fitness), one from each run. The analysis of the results makes it clear that F-DENSER++ generates higher performing CNNs than F-DENSER; the variation from the validation to the test set is small, and thus the networks generalise well.

The significance of the results is tested resorting to statistics. To understand if the samples follow a normal distribution we apply the Kolmogorov-Smirnov and Shapiro-Wilk tests ($\alpha = 0.05$). For all the collected data, these tests show that the data does not follow any distribution, and consequently to perform the pairwise comparison we use the Mann-Whitney U test ($\alpha = 0.05$). In addition, we measure the effect size: low ($0.1 \leq r < 0.3$), medium ($0.3 \leq r < 0.5$) and large ($r \geq 0.5$). The statistical tests show that the difference between F-DENSER++ and F-DENSER is statistically significant, with large effect sizes; the p-values are reported in Table 2.

In F-DENSER during evolution the networks are evaluated up to a maximum fixed time (10 minutes in the conducted experiments), and thus after evolution the best networks may benefit from re-training for longer. This is not required in F-DENSER++, because during evolution the time allocated for the training of each network can increase. The last row of Table 2 shows the results of F-DENSER, when the networks are re-trained until convergence (determined by early stopping). Even when re-trained the results of F-DENSER++

Table 2: Comparison of the results obtained on the evolution of the topology and learning strategy with F-DENSER++, and F-DENSER on the CIFAR-10. The results report the validation accuracy (fitness), and the test accuracy, and are measured with the generated networks right off evolution (evo), and when trained for longer (longer); the longer training is not applicable to F-DENSER++. The test (longer) results of F-DENSER are compared to the test (evo) results of F-DENSER++. Bold highlights statistically significant results.

	F-DENSER++	F-DENSER	p-value
Validation	89.44%	87.56%	**0.03752**
Test (evo)	88.73%	86.91%	**0.03156**
Test (longer)	n/a	87.76%	0.30772

are slightly superior to those of F-DENSER; however, the difference is not statistically significant.

To better analyse the results we use a box-plot (see Figure 4). The plot shows, as above stated, that F-DENSER++ evolutionary results are with no doubts superior to those of F-DENSER. On the other hand, it provides new insights on the comparison (over the test set) of F-DENSER++ with the re-trained networks of F-DENSER: despite not statistically different, the results of F-DENSER++ tend to be superior to the ones reported by F-DENSER – there are no outliers (despite the slightly larger dispersion), and the median of F-DENSER++ is above the median of F-DENSER; the difference in the median is of approximately 1%, which translates into about 100 more correctly labeled test instances.

From the above, it is demonstrated that F-DENSER++ can effectively generate networks that are ready to be deployed right-off evolution, i.e., there is no need for further training. This helps in the testing of the evolved training policy, as it is used until convergence; the training policies that are generated for F-DENSER despite providing good results when applied for longer training cycles may not be the most adequate ones. Most importantly, the above results are achieved without a major increase in the time required to search for the networks: from an average of 0.73 hours/generation to an average of 1.13 hours/generation, which is still fairly bellow the average of 10.83 hours/generation of DENSER[2]. From this point onward we focus on the use of F-DENSER++.

5 CONCLUSIONS

The current work introduces F-DENSER++: an extension to F-DENSER that enables it to generate fully-trained models, i.e., models that can be deployed right-off evolution. The results demonstrate that the evolutionary results of F-DENSER++ are statistically superior to those of F-DENSER. The results of F-DENSER still need to be trained for longer after evolution; nonetheless, the performance of the longer trains is still bellow the evolutionary performance of F-DENSER++. In addition, we can state that the new method is superior to the standard DENSER implementation; the evolutionary results of F-DENSER are statistically superior to DENSER, and F-DENSER++ is statistically superior to F-DENSER, and consequently superior to DENSER.

[2]All the times are measure in machines with the same specifications: 1080 Ti GPUs, 64 GB of RAM, and an Intel Core i7-6850K CPU.

Future work will be guided into two separate directions: (i) perform experiments with a wider set of datasets, and (ii) investigate transfer and multi-task learning with F-DENSER++. The common approach to NE seeks to generate a network for a specific task, without using any of the information gathered when addressing previous tasks. In the future, it is our objective to evolve DENSER to a point where learning is incremental and cumulative, using past knowledge, and avoiding catastrophic forgetting. That is, we want a system that grows with time, and learns new tasks without stopping being able to solve the previous ones.

ACKNOWLEDGMENTS

The work is partially supported by the Portuguese Foundation for Science and Technology under Grant No.: SFRH/BD/114865/2016.

REFERENCES

[1] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. 2018. DENSER: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines* (27 Sep 2018). https://doi.org/10.1007/s10710-018-9339-y

[2] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. 2019. Fast DENSER: Efficient Deep NeuroEvolution. In *European Conference on Genetic Programming*. Springer, 197–212.

[3] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 1 (01 Mar 2008), 47–62. https://doi.org/10.1007/s12065-007-0002-4

[4] Faustino J. Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. 2008. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *Journal of Machine Learning Research* 9 (2008), 937–965.

[5] Frédéric Gruau, Darrell Whitley, and Larry Pyeatt. 1996. A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks. In *Proceedings of the 1st Annual Conference on Genetic Programming*. MIT Press, Cambridge, MA, USA, 81–89. http://dl.acm.org/citation.cfm?id=1595536.1595547

[6] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[7] Pablo Ribalta Lorenzo and Jakub Nalepa. 2018. Memetic evolution of deep neural networks. In *GECCO*. ACM, 505–512.

[8] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sánchez Ramos, and José Ranilla Pastor. 2017. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *GECCO*. ACM, 481–488.

[9] Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. 2017. Evolving Deep Neural Networks. *CoRR* abs/1703.00548 (2017).

[10] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. 1989. Designing Neural Networks using Genetic Algorithms. In *ICGA*. Morgan Kaufmann, 379–384.

[11] David E Moriarty and Risto Miikkulainen. 2001. Learning sequential decision tasks through symbiotic evolution of neural networks. *Advances in the Evolutionary Synthesis of Intelligent Agents* (2001), 367.

[12] Gregory Morse and Kenneth O. Stanley. 2016. Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks. In *GECCO*. ACM, 477–484.

[13] José Parra, Leonardo Trujillo, and Patricia Melin. 2014. Hybrid back-propagation training with evolutionary strategies. *Soft Computing* 18, 8 (01 Aug 2014), 1603–1614. https://doi.org/10.1007/s00500-013-1166-8

[14] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548* (2018).

[15] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. 2017. Large-Scale Evolution of Image Classifiers. In *ICML (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, 2902–2911.

[16] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. 2009. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life* 15, 2 (2009), 185–212.

[17] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks Through Augmenting Topologies. *Evol. Comput.* 10, 2 (June 2002), 99–127. https://doi.org/10.1162/106365602320169811

[18] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *GECCO*. ACM, 497–504.

[19] Andrew James Turner and Julian Francis Miller. 2013. Cartesian genetic programming encoded artificial neural networks: a comparison using three benchmarks. In *GECCO*. ACM, 1005–1012.

[20] Darrell Whitley. 1989. Applying genetic algorithms to neural network learning. In *Proceedings of the Seventh Conference (AISB89) on Artificial Intelligence and Simulation of Behaviour*. Morgan Kaufmann Publishers Inc., 137–144.