SoftwareX 14 (2021) 100694

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

# Fast-DENSER: Fast Deep Evolutionary Network Structured Representation

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

*University of Coimbra, Centre for Informatics and Systems of the University of Coimbra, Department of Informatics Engineering, Portugal*

ARTICLE INFO

*Article history:*
Received 28 May 2019
Received in revised form 30 September 2020
Accepted 1 April 2021

*Keywords:*
Artificial Neural Networks
Automated machine learning
NeuroEvolution

ABSTRACT

This paper introduces a grammar-based general purpose framework for the automatic search and deployment of potentially Deep Artificial Neural Networks (DANNs). The approach is known as Fast Deep Evolutionary Network Structured Representation (Fast-DENSER) and is capable of simultaneously optimising the topology, learning strategy and any other required hyper-parameters (e.g., data pre-processing or augmentation). Fast-DENSER has been successfully applied to numerous object recognition tasks, with the generation of Convolutional Neural Networks (CNNs). The code is developed and tested in Python3, and made available as a library. A simple and easy to follow example is described for the automatic search of CNNs for the Fashion-MNIST benchmark.



## Code metadata

| | |
|---|---|
| Current code version | v2.1.0 |
| Permanent link to code/repository used for this code version | https://github.com/ElsevierSoftwareX/SOFTX_2019_192 |
| Legal Code License | Apache License, 2.0 |
| Code versioning system used | git |
| Software code languages, tools, and services used | Python3.7 |
| Compilation requirements, operating environments & dependencies | CUDA $\geq$ 10, CuDNN $\geq$ 7, tensorflow $\geq$ 2.0 (with GPU support), keras, scipy, sklearn, jsmin, Pillow |
| If available Link to developer documentation/manual | https://github.com/fillassuncao/fast-denser3 |
| Support email for questions | fga@dei.uc.pt |

## 1. Motivation and significance

The normal approach to the use of Artificial Neural Network (ANN) involves a cyclic manual trial-and-error process where the user optimises the: (i) topology of the network, i.e., type, position, and hyper-parameters of each layer; and (ii) the learning algorithm and its hyper-parameters. The problem is that the topology and learning strategy are not independent from each other. Moreover, when we require DANNs, the number of optimisable hyper-parameters can easily reach thousands or even millions.

To overcome the difficulty of deploying DANNs we can resort to NeuroEvolution (NE): a set of methods that apply Evolutionary Computation (EC) to the automatic optimisation of ANNs. The current paper focuses on a NE approach known as Fast-DENSER [1,2]: a variant of Deep Evolutionary Network Structured Representation (DENSER) [3]. The main advantage of Fast-DENSER over other NE methods is that not only it can generate high performing networks, but it does so in a fraction of the time, and using significantly less computational resources. Compared to the standard DENSER implementation, Fast-DENSER reports a speedup of 20x, without compromising the performance of the found solutions. In addition, Fast-DENSER is easy to use by a non-expert user: all the parameters and settings are defined in a text human-readable format. The software is flexible and easy to extend, giving the user the possibility to define new layers and/or operators.

The tool described in this paper has been successfully applied to automate the search for CNNs for object recognition tasks [1,2], where it generated networks that surpass or are competitive with the performance of other automatic methods, and that surpass the performance of human-designed models. Experiments have

* Corresponding author.
*E-mail addresses:* fga@dei.uc.pt (Filipe Assunção), naml@dei.uc.pt (Nuno Lourenço), bribeiro@dei.uc.pt (Bernardete Ribeiro), machado@dei.uc.pt (Penousal Machado).

https://doi.org/10.1016/j.softx.2021.100694

also been conducted in a problem from the physics domain, with the goal to distinguish between ground impact patterns of gamma and proton radiations [4]. The generated networks surpassed by a factor of 2 the performance of previous classic statistical methods.

## 2. Software description

The current section describes the architecture of the Fast-DENSER framework. The main objective and functionality of this library is to automate the search for DANNs. An example of the application of the framework to search for DANNs for a classification problem is described in Section 3.

The framework is based on EC, and thus a set of individuals (population) is evolved throughout a defined number of generations. The individuals encode DANNs and need to be mapped into interpretable models to assess their quality (fitness). To promote evolution from one generation to the next mutations are applied to the population. In particular, in Fast-DENSER the evolutionary engine is a (1+λ)-Evolutionary Strategy (ES): the next generation is formed by the best individual (elite), and λ mutations of it. For an in-depth description of the evolutionary search procedure refer to [1]

To map the individuals to interpretable models we resort to Keras [5] with Tensorflow [6] background. The framework requires the definition of two major inputs: (i) the network structure that establishes the enabled sequence of evolutionary units; and (ii) the grammar that defines the search space, i.e., layers and parameters. These two components are further detailed in the upcoming sub-sections. There are additional parameters that are enumerated in the framework GitHub page, which can be found at https://github.com/fillassuncao/fast-denser3. Fig. 1 shows the interaction between the evolutionary cycle and the inputs, and the mapping from the individuals into trainable DANNs for quality assessment.

The output of the framework is a fully-trained DANN, tailored to the considered problem. The network is made available as a Keras/Tensorflow model, but there is also a file specifying the structure and all network parameters (including weights) so that the user can later deploy the model in any other framework of his/her choice. Intermediate files are also generated throughout generations. In particular, there is a file for each generation; this file, among other properties, reports the phenotype (i.e., the actual network), fitness value, number of trainable parameters, and training time of each of the individuals of the population.

### 2.1. Network structure

The network structure is divided into 3 parts: (i) the hidden-layers; (ii) the output; and (iii) the macro blocks. The first part (hidden-layers) sets the sequence of evolutionary units that the framework can use to build DANNs, and is defined by the user as an ordered list of evolutionary units where each position stores the grammar non-terminal symbols, and the minimum and maximum number of evolutionary units of that type. The output sets the rule that should be used to form the output layer. Finally, the macro blocks consider the overall settings of the network, such as, the learning strategy to be used or the data pre-processing or augmentation policies. Together, the network structure and the grammar define the search space. An example of the hidden-layers, output, and macro blocks is respectively, [(features, 1, 10), (classification, 1, 10)]; softmax; and [learning]. The non-terminals require a one-to-one mapping to the grammar (discussed next). That is, the formed networks can have between 1 and 10 feature-related layers, and between 1 and 10 classification-related layers, that are followed by a softmax layer. The learning strategy is also evolved.

### 2.2. Grammar

In Fast-DENSER the domain is defined by means of a Backus–Naur form (BNF) grammar that encodes the hyper-parameters of each evolutionary unit. A grammar can be formally defined by a 4-tuple: $G = (N, T, P, S)$, where (i) $N$ is the set of non-terminal symbols; (ii) $T$ is the set of terminal symbols; (iii) $P$ is the set of production rules of the form $x ::= y$, $x \in N$ and $y \in N \cup T^*$; and (iv) $S$ is the initial symbol. The | denotes different possibilities for the expansion of a non-terminal symbol. A partial example of a grammar is shown in Fig. 2; the complete grammar can be found in https://github.com/fillassuncao/fast-denser3/blob/master/example/cnn.grammar. From the example, it is perceivable that for the expansion of the features non-terminal symbol, the convolution and pooling layers are repeated twice. This choice was made because all the expansion possibilities have the same probability of being selected. However, we want to bias the decision towards convolution and pooling layers. The non-terminals of the network structure are used as starting symbols for the encoding of the structure of the networks. The hyper-parameters can be either integer, float, or closed choice. The integer and float are parameterised using the block [parameter_name, type, num_values, min_value, max_value] (e.g., num-filters in line 4); the closed choice parameters are parameterised using the grammatical expansions (e.g., padding in line 7).

The evolutionary units encoding layers must start by layer: layer_type (e.g., layer:fc in line 12); the evolutionary units encoding the learning strategy must start by learning:learning_algorithm (e.g., learning:gradient-descent in line 16). Currently, we support the following layers: convolutional (conv), max-pooling (max-pool), average-pooling (avg-pool), fully-connected (fc), dropout (drop), and batch normalisation (batch-norm); and the following learning algorithms: gradient-descent, rmsprop, and adam. Nonetheless, to extend the framework to deal with other layer types, and/or learning algorithms, the user just needs to add the mapping code in the utils.py file /assemble_network and/or assemble_optimiser functions. An example of the extension to new layers is shown in the GitHub' page.

## 3. Illustrative example

To illustrate the functionalities of the framework we will address the evolution of CNNs for the Fashion-MNIST [7]: a dataset composed by greyscale fashion items of 10 independent classes. To promote the evolution of CNNs for the Fashion-MNIST dataset we use the grammar of Fig. 2, the outer-level structure [(features, 1, 30), (classification, 1, 10)], with the softmax to encode the output. The learning strategy is also evolved (learning production rule of the grammar).

To initialise the search we execute:

```
python –m fast_denser.engine –d fashion–mnist
                  –c config.json –g cnn.grammar
```

where -d, -c, and -g respectively set the dataset, and the paths to the configuration and grammar files. The network structure is one of the parameters of the configuration file. There is another optional input parameter, -r, that sets which run we want to perform (defaults to 0). Docker image files are available at https://hub.docker.com/r/fillassuncao/f-denser.

The intermediate evolution files with statistics on each generation are stored by default in a folder called experiments (set in the config.json file), where there is a sub-folder for each run. Each file keeps information about the unique identifier of the individual, phenotype, fitness value, metrics (e.g., training and validation loss and accuracy), number of trainable parameters, number of performed training epochs, maximum allowed training

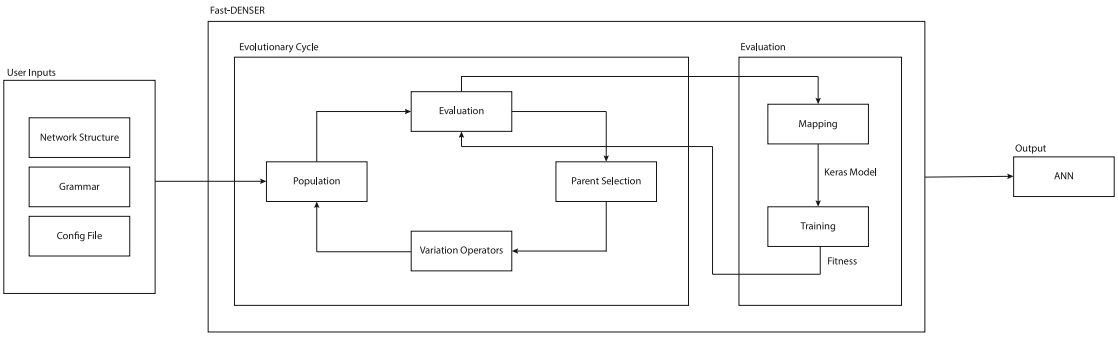


**Fig. 1.** Architecture of the framework.

```
<features> ::= <convolution> | <convolution>                          (1)
             | <pooling> | <pooling>                                   (2)
             | <dropout> | <batch-norm>                               (3)
<convolution> ::= layer:conv [num-filters,int,1,32,256]               (4)
             [filter-shape,int,1,2,5] [stride,int,1,1,3]              (5)
             <padding> <activation> <bias>                            (6)
<padding> ::= padding:same | padding:valid                            (7)
<classification> ::= <fully-connected> | <dropout>                    (8)
<fully-connected> ::= layer:fc <activation>                           (9)
             [num-units,int,1,128,2048 <bias>                         (10)
<bias> ::= bias:True | bias:False                                     (11)
<softmax> ::= layer:fc act:softmax num-units:2 bias:True              (12)
<learning> ::= <bp> <stop> [batch_size,int,1,50,300]                  (13)
             | <rmsprop> <stop> [batch_size,int,1,50,300]              (14)
             | <adam> <stop> [batch_size,int,1,50,300]                 (15)
<bp> ::= learning:gradient-descent [lr,float,1,0.0001,0.1]            (16)
             [momentum,float,1,0.68,0.99]                             (17)
             [decay,float,1,0.000001,0.001] <nesterov>                (18)
<stop> ::= [early_stop,int,1,5,20]                                    (19)
```

**Fig. 2.** Example of a grammar for encoding CNNs.

time, and performed training time; the files are formatted in JSON. The best individual found so far is stored in the best.h5 file – a model that can be loaded to Keras using the following code (also in Python):

```
from keras.models import load_model
model = load_model('best.h5')
```

which loads the topology and weights of the best generated model. New instances can be labelled using the predict method, i.e.:

```
import numpy as np
label_confidences = model.predict(instance)
label = np.argmax(label_confidences)
```

## 4. Impact

Fast-DENSER is a framework that promotes the automatic generation of DANNs, and thus avoids the user the burden of having to manually optimise a network that can solve a specific problem. Therefore it enables non-expert users to consider ANNs in their domains, and helps expert users tuning their networks and obtaining solutions that they would usually do not think of.

The framework is easy to use: all the parameters are defined in a human-readable format, and the output is a fully-trained DANN that can be deployed right-off evolution. This is an advantage comparing to the majority of other NE frameworks; they tend to evaluate the candidate solutions for a limited amount of time, and thus require further training by the end of the evolutionary search. This is a barrier to non-expert users. In addition, according to Baldominos et al. [8], Fast-DENSER uses an interesting representation scheme, and is categorised as a settlement approach, i.e., a stable work that has proved to effectively evolve DANNs.

Despite considering only a set of layers and learning algorithms the framework can be easily extended. The core is kept the same independently of the considered evolutionary units. The user just needs to add the code to map between the grammar and the Keras model, which is a simple parsing routine, similar to the mappings that are already in the code. Examples on how to extend the available layers and evaluation metrics are available in the GitHub's readme.

The framework has been widely tested and debugged, and has led to the generation of deep networks to numerous object recognition benchmarks [1,3]. Further, in the physics domain it has helped finding models that improve by a factor of 2 the gamma/hadron detection based on the ground impact [4]. In [3] we demonstrate that our approach is competitive with other

automatic (evolutionary and non-evolutionary) state-of-the-art methods, and in some cases even surpasses the performance of the best networks generated by the state-of-the-art approaches. A comparison with hand-designed networks is also conducted, once again showing the advantage of automation.

## 5. Conclusions

This paper describes the Fast-DENSER framework: a general-purpose tool for the automatic generation of DANNs of different structure, and to different problems. To adapt the method to different domains and network topologies the user is just required to change the network structure, and the grammar — the two variables that set the search space. An illustrative example on the evolution of CNNs for the Fashion-MNIST dataset is presented.

Future work will consider the expansion of the framework to address incremental and multi-task learning, i.e., it is our objective to help the user to cumulatively learn to solve new tasks but, without forgetting how to solve the previous ones. We will also integrate cloud computing platforms (e.g., Microsoft Azure, or Amazon AWS), to enable those users that do not have access to Graphics Processing Units (GPUs) to perform the trains on cloud services, and consequently speedup evolution. The first experiments on the expansion to incremental learning indicate that it is possible to speedup the search procedure, without compromising the end performance [9].

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

This work is funded by national funds through the FCT — Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/ 00326/2020 and under the Grant No.: SFRH/BD/114865/2016, and by European Social Fund, through the Regional Operational Program Centro 2020.

## References

[1] Assunção F, Lourenço N, Machado P, Ribeiro B. Fast DENSER: Efficient deep neuroevolution. In: European conference on genetic programming. Springer; 2019, p. 197–212.
[2] Assunção F, Lourenço N, Machado P, Ribeiro B. Fast-DENSER++: Evolving fully-trained deep artificial neural networks. 2019, arXiv preprint arXiv:1905.02969.
[3] Assunção F, Lourenço N, Machado P, Ribeiro B. DENSER: Deep evolutionary network structured representation. Genet Program Evolvable Mach 2019;20(1):5–35. http://dx.doi.org/10.1007/s10710-018-9339-y.
[4] Assunção F, Correia J, Conceição R, Pimenta MJM, Tomé B, Lourenço N, Machado P. Automatic design of artificial neural networks for gamma-ray detection. IEEE Access 2019;7:110531–40. http://dx.doi.org/10.1109/ACCESS.2019.2933947.
[5] Chollet F, et al. Keras. 2015, https://keras.io.
[6] Abadi M, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.
[7] Xiao H, Rasul K, Vollgraf R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. 2017, arXiv:cs.LG/1708.07747.
[8] Baldominos A, Saez Y, Isasi P. On the automated, evolutionary design of neural networks: Past, present, and future. Neural Comput Appl 2019;1–27.
[9] Assunção F, Lourenço N, Ribeiro B, Machado P. Incremental evolution and development of deep artificial neural networks. In: Hu T, Lourenço N, Medvet E, Divina F, editors. Genetic programming. Cham: Springer International Publishing; 2020, p. 35–51.