

Towards Physical Plausibility in Neuroevolution Systems

Gabriel Cortês^[0000-0001-6318-8520], Nuno Lourenço^[0000-0002-2154-0642], and
Penousal Machado^[0000-0002-6308-6484]

University of Coimbra, CISUC/LASI – Centre for Informatics and Systems of the
University of Coimbra, Department of Informatics Engineering
{cortes,naml,machado}@dei.uc.pt

Abstract. The increasing usage of Artificial Intelligence (AI) models, especially Deep Neural Networks (DNNs), is increasing the power consumption during training and inference, posing environmental concerns and driving the need for more energy-efficient algorithms and hardware solutions. This work addresses the growing energy consumption problem in Machine Learning (ML), particularly during the inference phase. Even a slight reduction in power usage can lead to significant energy savings, benefiting users, companies, and the environment. Our approach focuses on maximizing the accuracy of Artificial Neural Network (ANN) models using a neuroevolutionary framework whilst minimizing their power consumption. To do so, power consumption is considered in the fitness function. We introduce a new mutation strategy that stochastically reintroduces modules of layers, with power-efficient modules having a higher chance of being chosen. We introduce a novel technique that allows training two separate models in a single training step whilst promoting one of them to be more power efficient than the other while maintaining similar accuracy. The results demonstrate a reduction in power consumption of ANN models by up to 29.2% without a significant decrease in predictive performance.

Keywords: Evolutionary Computation · Neuroevolution · Energy Efficiency

1 Introduction

As the demand for Machine Learning (ML) continues to grow, so does the electrical power required for training and assessment. According to Patterson et al., GPT-3, the model behind ChatGPT, consumes 1287 MWh, corresponding to approximately 552 tons of CO₂ equivalent emissions just for training during 15 days [16]. In addition to the environmental impacts of this power usage, it can also burden individual users and organizations, who may face high energy costs. Therefore, finding ways to reduce the power consumption of ML processes is becoming increasingly important.

Artificial Neural Networks (ANNs) are a type of ML model inspired by biological neural networks [19]. They consist of multiple layers of artificial neurons,

which are functions that take input data and produce an output based on it. The connections between neurons have an associated weight value modified in the training process to allow the network to “learn” how to solve a specific task. Deep Neural Networks (DNNs) are ANNs with a considerable number of hidden layers [9,10]. This allows them to avoid the feature engineering step, thus automatically discovering the representations needed for classification and achieving higher accuracy values. Training and executing ANNs is power-intensive due to the required computational resources.

Evolutionary Algorithms (EAs) are algorithms inspired by natural selection [6,17]. To evolve solutions over multiple generations, they utilize mechanisms, such as selection, crossover, and mutation. The process begins with a randomly initialized population whose evolution is steered by a fitness function that measures the quality of an individual. In conjunction with the mentioned evolutionary mechanisms, the process is predicted to culminate in near-optimal individuals.

Neuroevolution (NE) uses EAs to generate and optimize ANNs for a given task [7]. It can optimize the ANN’s architecture and hyperparameters.

We hypothesise that we can address the energy inefficiency issue by using NE to search for well-suited models for a particular problem while being power-efficient. Fast Deep Evolutionary Network Structured Representation (Fast-DENSER) is a method that utilizes an Evolution Strategy (ES) to find optimal ANN models by using their accuracy as the fitness function, thus guiding the search towards accurate models [2].

In this work, we propose novel approaches integrated into Fast-DENSER to find power-efficient models. We have incorporated a new approach to measure the power consumption of a DNN model during the inference phase. This metric has been embedded into multi-objective fitness functions to steer the evolution towards more power-efficient DNN models. We also introduce a new mutation strategy that allows the reutilization of modules of layers with inverse probability to the power usage of a module, thus (re)introducing efficient sets of layers in a model. We propose the introduction of an additional output layer connected to an intermediate layer of a DNN model and posterior partitioning into two separate models to obtain smaller but similarly accurate models that utilize less power. To the best of our knowledge, no prior works employ a similar approach.

The experiments are analyzed through two metrics: accuracy and mean power usage during the validation step. The motive for using the power usage of the validation step instead of the training step is that the training is usually performed only once. Contrarily, the inference is executed multiple times. Moreover, inference does not necessarily occur on the machine where the training was conducted, which is vital since many devices are not optimized for these tasks.

The results of this work show that it is possible to have DNN models with substantially inferior power usage. The best model found regarding power consumes 29.18 W (29.2%) less whilst having a tiny decrease in performance (less than 1%).

This work is structured as follows: Section 2 provides background information on ANNs, and NE. Section 3 introduces our methodologies to enhance the power efficiency of ANN models. Section 4 outlines the experimental setup. Section 5 presents the experimental results. Finally, in Section 6, we provide our conclusions and prospects for future research.

2 Background

2.1 Artificial Neural Networks

Artificial Neural Networks are a type of supervised ML inspired by biologic neural networks [19]. An ANN consists of connected processing units known as neurons. The connections follow a specific topology to achieve the desired application. A neuron’s input may be the output of other neurons, external sources, or itself. Every connection has an associated weight, allowing the system to simulate biological synapses. A weighted sum of the inputs is computed at a given instant, considering the connection weights. It is also possible to sum a bias value to this. An activation function is applied, and thus, the neuron’s output is obtained.

DNNs are ANNs composed of many hidden layers. Due to this, DNNs can avoid the feature engineering step – which usually requires human expertise – by automatically discovering the representations needed for classification [9,10]. Thus, they can model more complex relationships and achieve higher accuracy on tasks requiring pattern recognition. The development and usage of DNNs have substantially increased due to the widespread deployment of more capable hardware, such as Graphics Processing Units (GPUs) [3].

2.2 Neuroevolution

NE is the application of evolutionary techniques to search for DNN models. It is used to optimize the structure and weights of DNNs to improve their performance on specific tasks, such as image classification and natural language processing. NE is a gradient-free method based on the concept of population [7]. It allows for the simultaneous exploration of multiple zones of the search space through parallelization techniques at the cost of taking a usually long time to execute since each individual of the population is a DNN that requires training and testing.

Deep Evolutionary Network Structured Evolution (DENSER) is a neuroevolutionary framework that allows the search of DNNs through a grammar-based neuroevolutionary approach that searches both network topology and hyperparameters [1].

The developed DNNs are structured according to a provided context-free grammar. DENSER uses Dynamic Structured Grammatical Evolution (DSGE) as the strategy that allows the modification of the network topology. DSGE is built upon Structured Grammatical Evolution (SGE), with the main differences

of allowing the growth of the genotype and only storing encoded genes [11]. Allied with dynamic production rules, DSGE allows the creation of multiple-layer DNNs. SGE proves to perform better than Grammatical Evolution (GE), and DSGE proves to be superior to SGE [12]. The individuals of the evolutionary process are represented in two levels: the outer level encodes the topology of the ANN, and the inner one encodes its hyperparameters.

Fast-DENSER was developed to overcome some limitations verified on DENSER: evaluating the population consumes a considerable amount of time, and the developed DNNs are not fully trained [2]. Fast-DENSER is an extension of DENSER on which the evolutionary engine is replaced by a $(1 + \lambda)$ -ES. This modification dramatically reduces the required number of evaluations per generation, enabling executions 20 times faster than the original version of DENSER.

Moreover, individuals are initialized with shallow topologies, and the stopping criterion is variable to allow an individual to be trained for a more extended time.

On the CIFAR-10 dataset [8], DENSER obtained models with an accuracy higher than most of the state-of-the-art results, and on the CIFAR-100 [8], it obtained the best accuracy reported by NE approaches. Fast-DENSER proves to be highly competitive relative to DENSER, achieving execution times far inferior to its predecessor. Additionally, Fast-DENSER can develop DNNs that do not require additional training after the evolutionary approach and are, therefore, ready to be deployed.

3 Approach

This section outlines the approaches developed to address the challenge of reducing power consumption in ANN models.

3.1 Power Measurement

Measuring the power a GPU consumes is fundamental when developing approaches that minimize a model’s energetic footprint. The ecosystem of developing a DNN model mainly consists of three phases: design, training, and deployment.

The design phase uses some energy, be it with manual design techniques or automatic methods. DENSER is a NE framework and, as such, consumes energy in the search for optimal models, and such consumption might be on par with the energy used on manual, trial-and-error methods. Reducing the energy used in this phase is out of the scope of this work.

The training of a DNN model is an expensive process in which a model is trained on a large dataset to learn to predict unseen instances, taking a significant toll on technological companies’ and individuals’ power bills. While diminishing energy consumption during the training process remains a significant objective, it is worth noting that the inference phase in DNNs holds vital importance during software deployment, as the software obtains results through inference.

This becomes particularly relevant when considering the potential utilization of these models by millions of users. As such, tackling the minimization of energy consumption in this step is vital. For example, it is estimated that 80% to 90% of NVIDIA’s ML computations are inference processing [13] and about 60% of Google’s ML energy usage is for inference with the remaining portion being for training [16].

Considering this, our work focuses on the power consumption in the inference step to allow a large deployment, thus saving more computational resources and energy and, on another layer, reducing financial expenses and reducing environmental impact.

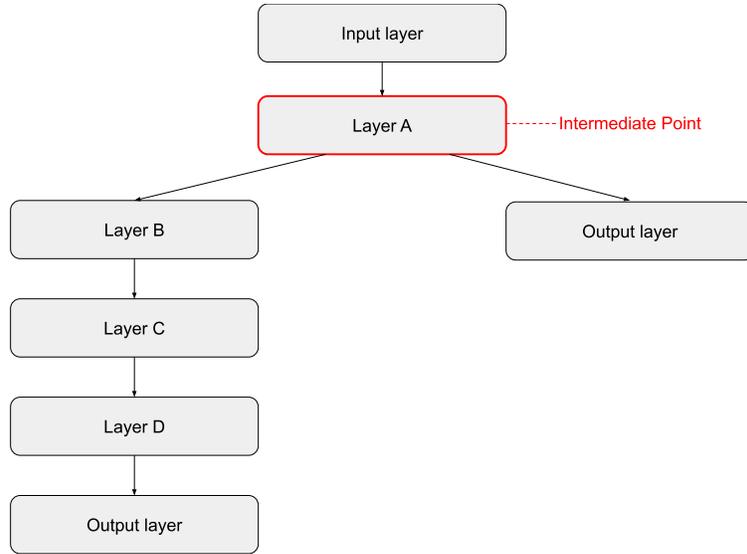
3.2 Model Partitioning

Training a DNN model requires a substantial amount of time and considerable energy. Creating a process on which a single model is trained but can be split posteriorly into two models would reduce the time spent on training two models by, at most, two times. Pushing one of those two models into being smaller than the other may produce a simpler, similarly accurate, yet more power-efficient model.

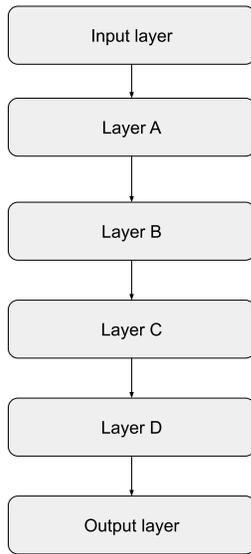
Following this line of reasoning, we propose a modification to Fast-DENSER on which an extra output layer is connected to an intermediate layer of the model. The two-output model (Figure 1a) is trained to optimize for two outputs. At the validation step, it is split into left (Figure 1b) and right (Figure 1c) partitions. These partitions are disjoint and can be evaluated similarly to how the complete model is evaluated, and metrics such as accuracy and power consumption can be obtained.

The intermediate point is a marker for where the additional output is added at the model partitioning step. We can, for example, consider a model as an array of layers, and the mentioned marker is the index of the layer to which the additional output is connected. This point can be assigned to any intermediate layer of the model. The input and output layers are excluded to prevent useless and redundant partitions.

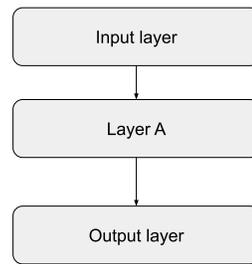
Since the maximum allowed value of the point is equal to the number of layers of the model minus one, the grammar initializer – which generates individuals according to the grammar – and the mutation mechanism for the macrostructure level of DENSER – which performs mutations on the hyperparameters of the individuals – were modified to consider the maximum number of layers of the model dynamically. To introduce the intermediate point in the evolutionary process, it was considered part of the macrostructure and, as such, as a rule of the grammar. The introduced rule is $\langle middle_point \rangle ::= [middle_point, int, 1, 0, x]$, meaning that one integer value is obtained with the lower limit being zero. The upper limit is an arbitrary variable x that is replaced at any instance by the maximum number of layers of the model minus one.



(a) Full model



(b) Left partition



(c) Right partition

Fig. 1: Example of a two-output model and its left and right partitions, with the layer marked by the intermediate point in red.

3.3 Fitness Functions

To consider accuracy and power consumption in the fitness function, some functions were developed to take these parameters into account. Since our objective is to maximize accuracy but minimize power consumption, we consider the inverse of the latter, i.e., $power^{-1}$.

Considering our approach of the division of a DNN model into two comparably accurate partitions, with one smaller than the other, all of the presented fitness functions consider the accuracy of both partitions, intending to enhance both. These fitness functions only focus on minimizing power consumption within the larger partition, which is anticipated to experience higher power usage.

Firstly, as presented in Equation 1, we developed a fitness function that sums the accuracy of both partitions with the inverse of the power usage of the left partition. The accuracy values have an upper limit, consisting of minimum satisfiable values for the models, i.e., values below the state-of-the-art [15] to allow some tradeoff between accuracy and power consumption. The upper limit is higher on the right partition (0.85) than on the left partition (0.80) since it is desired that the right partition obtains a higher accuracy value, if possible. The goal of this function design was to obtain satisfiable models and, after that, guide the evolutionary process only by their power usage to minimize the power usage of the models. After testing, we observed that the power usage typically falls within the range [30, 100] W, which, when inverted, resulted in values too small to be able to properly steer the evolutionary process.

$$f_1 = \min(0.80, acc_{left}) + \min(0.85, acc_{right}) + power_{left}^{-1} \quad (1)$$

Considering this, another fitness function was designed (Equation 2), where the power usage is multiplied by 10, thus giving it a more considerable weight since power usage values for the used GPU typically fall within the [30, 100] W range. This weight is closely related to the used GPU and should be modified accordingly. Preliminary experiments showed that although the evolution managed to somewhat minimize the power usage of the models, their accuracy remained around the chosen upper limits. Since this is not an optimal behaviour, a function that does not limit accuracy was developed.

$$f_2 = \min(0.80, acc_{left}) + \min(0.85, acc_{right}) + 10 * power_{left}^{-1} \quad (2)$$

As shown in Equation 3, this fitness function considers only the accuracy of the partitions when both are below a threshold. After any of them surpass their respective threshold, power consumption is also considered, with a weight of 10. This means that, at first, evolution is only steered by the accuracy of the models. When satisfiable models are obtained, power consumption starts being considered to evolve both accurate and energy-efficient models.

$$f_3 = \begin{cases} acc_{left} + acc_{right} & \text{if } acc_{left} \leq 0.80 \wedge acc_{right} \leq 0.85 \\ acc_{left} + acc_{right} + 10 * power_{left}^{-1}, & \text{otherwise} \end{cases} \quad (3)$$

3.4 Module Reutilization

Internally, Fast-DENSER considers modules of layers on each individual from which a DNN is then unravelled. One way to encourage the evolution of energy-efficient models is to provide an individual with a set of layers that are known to be efficient. As such, a scheme of module reutilization is proposed through the design of new mutation operators and the addition of an archive of modules and their respective power consumption.

Since this strategy only considers power consumption, it is expected that inaccurate models may sometimes be generated. Due to the nature of the evolutionary process and the used fitness function (Equation 3), inaccurate models are intensely penalized and, as such, discarded in favour of better ones.

Whenever a module of layers is randomly generated or modified, its power consumption is measured. To do this, a temporary model is created, which consists of an input layer, the module’s layers, and an output layer. Since the module’s accuracy is irrelevant, this temporary network is neither trained nor fed with a proper dataset, i.e., it is given random values instead of a dataset.

An operator of mutation, *reuse module*, was introduced to take advantage of this information. It selects a module with a probability inversely proportional to its power consumption, i.e., modules with inferior power consumption have a superior probability of being chosen. As shown in Equation 4, to obtain the probability of a module i being chosen, we divide the inverse of its power, $power_i$, by the sum of the inverse power of all modules, with n the number of saved modules. The selected module is introduced in a randomly chosen position. An operator that randomly removes a module from an individual is also introduced to counteract the described operator.

$$P(i) = \frac{\frac{1}{power_i}}{\sum_{j=0}^n \frac{1}{power_j}} \quad (4)$$

4 Experimental Setup

We performed two experiments: the baseline, which uses the plain version of Fast-DENSER with accuracy as the fitness function, and an experiment where our proposed approaches were applied, using the fitness function presented in Equation 3. Table 1 presents the experimental parameters used across the experiments. Note that *DSGE-level rate* refers to the probability of a grammar mutation on the model’s layers, the *Macro layer rate* pertains to the probability of a grammar mutation affecting the macrostructure, encompassing elements such as hyperparameters or intermediate point mutation, and the *Train longer rate* is the probability of allocating more time for an individual to be trained. The rates of reusing and removing modules do not apply to the baseline experiment. The experimental analyses consider the Mean Best Fitness (MBF) over 5 runs.

The experiments were performed on a server running Ubuntu 20.04.3 LTS with an Intel Core i7-5930K CPU with a clock frequency of 3.50GHz, 32 GB

of RAM, and an NVIDIA TITAN Xp with CUDA 11.2, CuDNN 8.1.0, Python 3.10.9, Tensorflow 2.9.1 and Keras 2.9.0 installed as well as the pyJoules 0.5.1 Python module with the NVIDIA specialization.

Table 1: Experimental parameters

Evolutionary Parameter	Value
Number of runs	5
Number of generations	150
Maximum number of epochs	10 000 000
Population size	5
Add layer rate	25%
Reuse layer rate	15%
Remove layer rate	25%
Reuse module rate	15%
Remove module rate	25%
Add connection	0%
Remove connection	0%
DSGE-level rate	15%
Macro layer rate	30%
Train longer rate	20%
Train Parameter	Value
Default train time	10 min
Loss	Categorical Cross-entropy

All experiments used the Fashion-MNIST dataset [18], which was developed as a more challenging replacement for the well-known MNIST dataset [5] by swapping handwritten digits with images of clothes such as shirts and coats, aiming at a more realistic and relevant benchmark. It is a balanced dataset consisting of a collection of 60 thousand examples for training and 10 thousand for testing, where each example is a 28x28 grey-scale image representing clothing items belonging to one of ten classes.

Since power usage is essential in making NE physically plausible, a function to measure power was developed using the *pyJoules* library. Its pseudocode can be analyzed in Algorithm 1, with *meter* being the library tool that facilitates the measurement of energy consumed, and *start* and *stop* the functions that allow controlling it. It wraps a function call (*func*, with corresponding arguments *args*) while measuring the GPU energetic consumption during its execution and the call’s duration. This measurement is converted from milli-Joule to Watt and appended to the array of measures. These steps are performed $n_measures$ times, and then the mean value is calculated. In our work, we considered $n_measures = 30$. The described function was integrated with Fast-DENSER on the model’s validation step to measure the power used in the inference phase.

Algorithm 1 Power Measure Algorithm

```

Require: func, args, n_measures
measures  $\leftarrow \emptyset$ 
i  $\leftarrow 1$ 
while i  $\leq n\_measures$  do
  start(meter)
  output  $\leftarrow func(args)$ 
  stop(meter)
  (energy, duration)  $\leftarrow measure(meter)$ 
  measure  $\leftarrow energy/1000/duration$  ▷ Convert mJ to W
  measures  $\leftarrow measures \cup measure$ 
  i  $\leftarrow i + 1$ 
end while
mean_power  $\leftarrow mean(measures)$ 
return (output, mean_power)

```

It should be noted that ambient conditions of the server’s location, such as temperature and humidity, were not considered, as well as other external variables, and no other processes used the GPU during the execution of these experiments.

5 Results

This section compares the results from the baseline experiment and the experiment where our approaches were applied. The results show the mean accuracy and the mean power consumption, which are derived from the best individuals by fitness over 5 separate runs.

Since the results did not follow a normal distribution and the samples were independent, the Kruskal-Wallis non-parametric test was employed to determine if significant differences existed among the various approach groups. When significant differences were observed, the Mann-Whitney post-hoc test with Bonferroni correction was applied. We considered a significance level of $\alpha = 0.05$ in all statistical tests.

Figure 2 compares the accuracy obtained in the two experiments. The experiments present a similar accuracy until generation 70, where it becomes possible to observe a clear difference between them. The baseline experiment achieves a higher accuracy than the other experiment, and, relative to that experiment, it is visible that the smaller model obtains a marginally smaller accuracy than the larger one. Table 2 provides statistical analysis, and Table 3 showcases statistical values of the experiments. It is possible to see that, relative to the median values, the proposed method achieves inferior accuracy and that the smaller model obtains the worst accuracy.

Figure 3 presents a comparison of the power consumption measured in the two experiments. The baseline predominantly has an increasing behaviour, which

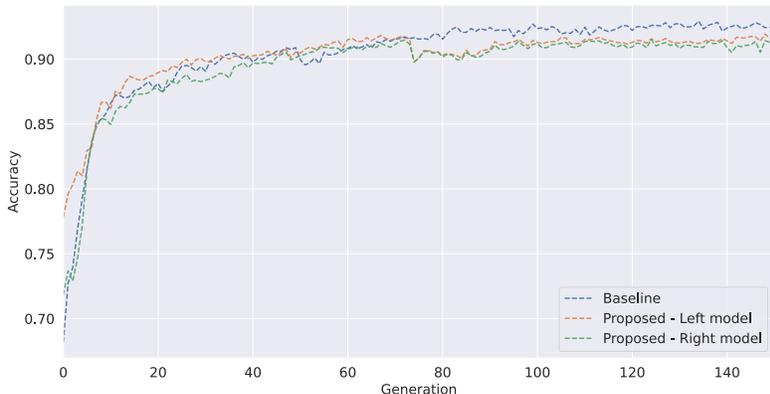


Fig. 2: Evolution of the accuracy over 150 generations.

Table 2: Pair-wise comparison of used groups on accuracy metric, using Mann-Whitney U post-hoc test with Bonferroni correction with bold values denoting statistically significant differences.

		Baseline	Proposed Method	
Metric		<i>Accuracy</i>	<i>Accuracy_{left}</i>	<i>Accuracy_{right}</i>
Baseline	<i>Accuracy</i>			
Proposed Method	<i>Accuracy_{left}</i>	1.09×10^{-4}		
	<i>Accuracy_{right}</i>	1.15×10^{-7}	1.16×10^{-4}	

can be explained by the fact that the evolution is only being guided by accuracy, i.e., there are no incentives to favour models that consume less power. Contrarily, the proposed method obtained relatively stable results over the evolutionary process, with the smaller model presenting marginally lower results than its counterpart. Table 4 provides statistical analysis, and Table 5 shows statistical values of the experiments. We can conclude that relative to the median values, the proposed method achieves inferior power consumption and that the smaller model is the most power-efficient.

6 Conclusion

In this work, we developed approaches integrated into Fast-DENSER, which empower it to generate DNN models with better power efficiency.

The most fundamental approach consists of measuring the power consumed by the GPU on the inference phase of the DNN. We use the measure provided

Table 3: Mean value, standard deviation, median and difference to baseline median of the accuracy of the experiments.

Experiment	Metric	Mean	SD	Median	Diff. to Baseline
Baseline	<i>Accuracy</i>	0.904	0.037	0.916	
Proposed Method	<i>Accuracy_{left}</i>	0.902	0.024	0.911	-0.005
	<i>Accuracy_{right}</i>	0.895	0.034	0.907	-0.009

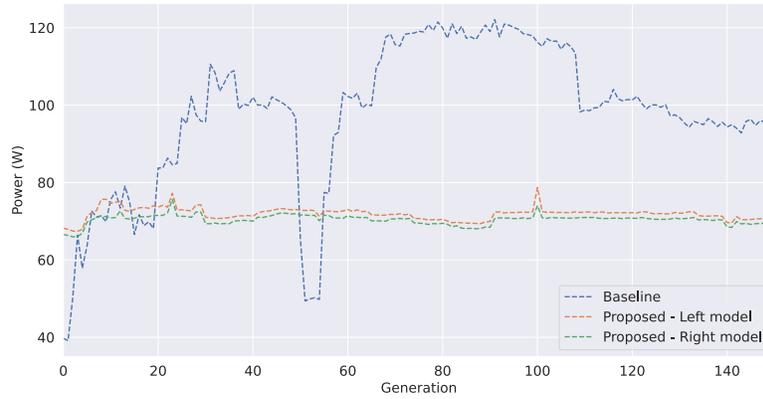


Fig. 3: Evolution of the power consumption over 150 generations.

by the GPU to do this. Using this metric, we developed multi-objective fitness functions that steer the evolutionary process in a path that minimizes power consumption.

We created a process by which an additional output is added to a DNN model and, after being trained, the model is split into two models – a larger one which consists of all the layers and a smaller one composed of the layers up to the one where the additional output is connected to. This allows us to create models tuned for environments with fewer resources, such as smartphones, while creating more power-intensive models tuned for environments with more resources, such as servers. This is performed in one training, thus taking less time to develop the two models and saving energy in the process. No prior work has been identified that employs a similar approach.

We introduced a new mutation strategy to Fast-DENSER that allows the reutilization of sets of layers – modules – according to the power consumption of the modules. We stochastically favour the reintroduction of modules in a model according to the inverse of the power they consume, thus incorporating power-efficient modules into a model.

Table 4: Pair-wise comparison of used groups on power metric, using Mann-Whitney U post-hoc test with Bonferroni correction with bold values denoting statistically significant differences.

		Baseline	Proposed Method	
Metric		$Power$	$Power_{left}$	$Power_{right}$
Baseline	$Power$			
Proposed Method	$Power_{left}$	2.72×10^{-29}		
	$Power_{right}$	8.84×10^{-32}	3.67×10^{-19}	

Table 5: Mean value, standard deviation, median and difference to baseline median of the experiments power consumption.

Experiment	Metric	Mean	SD	Median	Diff. to Baseline
Baseline	$Power$	97.80 W	18.84 W	99.89 W	
Proposed Method	$Power_{left}$	71.92 W	1.60 W	72.20 W	-27.69 W
	$Power_{right}$	70.40 W	1.30 W	70.71 W	-29.18 W

The results obtained by our proposals show that we can reduce the power consumption of the ANNs without compromising their predictive performance, showing that it is possible to minimize power consumption while, at the same time, maximizing accuracy through the usage of NE frameworks such as Fast-DENSER. The best model found regarding power consumes 29.18 W (29.2%) less whilst having a tiny decrease in performance (less than 1%), proving that a small trade-off on accuracy can yield a considerable reduction in the power consumed by the model.

6.1 Future Work

We introduced novel approaches and performed a baseline experiment and an experiment where the mentioned strategies were applied. It could be valuable to explore other approaches and perform more experiments in the future.

To better understand the individual impact of each strategy on the efficiency of the models, it would be valuable to perform experiments with the employment of only one strategy at a time. It would also be interesting to vary the fitness functions (e.g., the weights used in them) and to vary evolutionary parameters such as the probabilities of the mutations.

One of the most important constraints of our work is GPU-time due to the amount of operations required to train every model of each generation. To minimize the required time, it would be noteworthy to research how to employ training-less strategies in Fast-DENSER, i.e., use strategies that estimate the

accuracy of a model without training it [4,14]. Such strategies would allow us to perform more experiments in less time, saving energy in the design process.

Acknowledgments

This work was supported by the Portuguese Recovery and Resilience Plan (PRR) through project C645008882-00000055, Center for Responsible AI, by the FCT, I.P./MCTES through national funds (PIDDAC), by Project No. 7059 - Neuraspace - AI fights Space Debris, reference C644877546-00000020, supported by the RRP - Recovery and Resilience Plan and the European Next Generation EU Funds, following Notice No. 02/C05-i01/2022, Component 5 - Capitalization and Business Innovation - Mobilizing Agendas for Business Innovation, and within the scope of CISUC R&D Unit - UIDB/00326/2020.

References

1. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: DENSER: Deep evolutionary network structured representation. *Genet. Program. Evolvable Mach.* **20**(1), 5–35 (2019). <https://doi.org/10.1007/S10710-018-9339-Y>
2. Assunção, F., Lourenço, N., Ribeiro, B., Machado, P.: Fast-DENSER: Fast deep evolutionary network structured representation. *SoftwareX* **14**, 100694 (2021). <https://doi.org/10.1016/j.softx.2021.100694>
3. Balas, V., Roy, S., Sharma, D., Samui, P.: *Handbook of Deep Learning Applications, Smart Innovation, Systems and Technologies*, vol. 136. Springer, Netherlands, 1 edn. (2019). <https://doi.org/10.1007/978-3-030-11479-4>
4. Chen, W., Gong, X., Wang, Z.: Neural architecture search on ImageNet in four GPU hours: A theoretically inspired perspective. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021). <https://doi.org/10.48550/arXiv.2102.11535>
5. Deng, L.: The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **29**(6), 141–142 (2012). <https://doi.org/10.1109/MSP.2012.2211477>
6. Eiben, A.E., Smith, J.E.: *Introduction to Evolutionary Computing*. Springer, 2nd edn. (2015). <https://doi.org/10.1007/978-3-662-44874-8>
7. Galván, E., Mooney, P.: Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Trans. Artif. Intell.* **2**(6), 476–493 (2021). <https://doi.org/10.1109/TAI.2021.3067574>
8. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009), <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
9. LeCun, Y., Bengio, Y., Hinton, G.E.: Deep learning. *Nat.* **521**(7553), 436–444 (2015). <https://doi.org/10.1038/nature14539>
10. Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E.: A survey of deep neural network architectures and their applications. *Neurocomputing* **234**, 11–26 (2017). <https://doi.org/10.1016/J.NEUCOM.2016.12.038>
11. Lourenço, N., Assunção, F., Pereira, F.B., Costa, E., Machado, P.: Structured grammatical evolution: A dynamic approach. In: Ryan, C., O’Neill, M., Collins, J.J. (eds.) *Handbook of Grammatical Evolution*, pp. 137–161. Springer (2018). https://doi.org/10.1007/978-3-319-78717-6_6

12. Lourenço, N., Pereira, F.B., Costa, E.: SGE: A structured representation for grammatical evolution. In: Artificial Evolution. Lecture Notes in Computer Science, vol. 9554, pp. 136–148. Springer (2015). https://doi.org/10.1007/978-3-319-31471-6_11
13. Luccioni, A.S., Viguier, S., Ligozat, A.L.: Estimating the carbon footprint of BLOOM, a 176B parameter language model (2022). <https://doi.org/10.48550/ARXIV.2211.02001>
14. Mellor, J., Turner, J., Storkey, A.J., Crowley, E.J.: Neural architecture search without training. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 139, pp. 7588–7598. PMLR (2021). <https://doi.org/10.48550/arXiv.2006.04647>
15. Meshkini, K., Platos, J., Ghassemian, H.: An analysis of convolutional neural network for fashion images classification (Fashion-MNIST). In: Proceedings of the Fourth International Scientific Conference “Intelligent Information Technologies for Industry” (IITI’19). pp. 85–95. Springer (2020). https://doi.org/10.1007/978-3-030-50097-9_10
16. Patterson, D.A., Gonzalez, J., Hölzle, U., Le, Q.V., Liang, C., Munguia, L., Rothchild, D., So, D.R., Texier, M., Dean, J.: The carbon footprint of machine learning training will plateau, then shrink. *Computer* **55**(7), 18–28 (2022). <https://doi.org/10.1109/MC.2022.3148714>
17. Vikhar, P.A.: Evolutionary algorithms: A critical review and its future prospects. In: 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC). pp. 261–265 (2016). <https://doi.org/10.1109/ICGTSPICC.2016.7955308>
18. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a Novel image dataset for benchmarking machine learning algorithms (2017). <https://doi.org/10.48550/arXiv.1708.07747>
19. Yegnanarayana, B.: Artificial Neural Networks. PHI Learning (2009)