1 2 9 0

UNIVERSIDADE Ð COIMBRA

Gabriel Alexandre Rodrigues Cortês

Towards Physical Plausibility in Neuroevolution Systems

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, supervised by Professor Fernando Jorge Penousal Martins Machado and Professor Nuno António Marques Lourenço and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2023

This page is intentionally left blank.

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Đ
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Gabriel Alexandre Rodrigues Cortês

Towards Physical Plausibility in Neuroevolution Systems

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, supervised by Professor Fernando Jorge Penousal Martins Machado and Professor Nuno António Marques Lourenço and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2023

This page is intentionally left blank.

This work was developed in collaboration with:

CISUC
CENTRO DE INFORMÁTICA E SISTEMAS
DA UNIVERSIDADE DE COIMBRA

Center for
Responsible AI

This page is intentionally left blank.

The work presented in this thesis was carried out within the Cognitive and Media Systems Group (CMS) and Evolutionary and Complex Systems (ECOS) groups of the Centre for Informatics and Systems of the University of Coimbra (CISUC).

This research was supported by the Portuguese Recovery and Resilience Plan (PRR) through project C645008882-00000055, Center for Responsible AI.

This work has been partially supported by the following grants:

- Grant n. 820474/IT137-22-015.
- Grant n. 874294/IT137-23-058.

This page is intentionally left blank.

This page is intentionally left blank.

"Trade-offs," he said out loud.
"It's always trade-offs."

JAMES S. A. COREY

This page is intentionally left blank.

Abstract

The widespread and increasing usage of Machine Learning (ML) results in significant power consumption in training and inference steps. Even a marginal reduction in power consumption holds the potential for substantial energy savings, benefiting stakeholders (e.g., companies, end users) and the environment.

Optimizing Artificial Neural Network (ANN) models is a promising avenue for curbing power consumption. Employing an evolutionary approach to tailor these models to specific problems offers a versatile framework. Furthermore, by integrating considerations of model power usage into this evolutionary process, the pathway toward power-efficient models can be efficiently paved.

In this work, we propose novel approaches integrated into Fast Deep Evolutionary Network Structured Representation (Fast-DENSER), which aim at finding power-efficient models. We have incorporated a new approach to measure the power consumption of a Deep Neural Network (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 reuse 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. We developed a strategy that allows initializing Fast-DENSER with a previously trained model to start the evolution with already accurate models and evolve them towards power-efficient models.

The results obtained by our proposals show that we can reduce the power consumption of the ANNs without compromising their predictive performance. In concrete, we can obtain models that consume less 19.50 W (19.5%) than a baseline model whilst having an accuracy higher by 0.2%. The best model found regarding power consumes less 29.18 W (29.2%) whilst having a tiny decrease in performance (less than 1%).

Keywords

Neuroevolution, Artificial Neural Network, Deep Neural Network, Evolutionary Computation, Energy Efficiency, Machine Learning

Resumo

A utilização generalizada e crescente de Aprendizagem Computacional (AC) resulta num consumo significativo de energia nas etapas de treino e inferência. Uma redução apenas marginal no consumo energético tem o potencial de levar a poupanças energéticas substanciais, beneficiando assim tanto as partes interessadas (e.g., empresas, utilizadores) como o meio ambiente.

A otimização de modelos de Redes Neuronais Artificiais constitui uma abordagem promissora para a redução de consumo energético. A utilização de uma abordagem evolutiva para afinar esses modelos a problemas específicos oferece uma plataforma versátil. Além disso, ao integrar considerações sobre o consumo energético de um dado modelo no processo evolutivo, é possível pavimentar de forma eficiente o caminho no sentido de modelos energicamente eficientes.

Neste trabalho, propomos abordagens novas integradas no Fast Deep Evolutionary Network Structured Representation (Fast-DENSER), que visam encontrar modelos energicamente eficientes. Incorporámos uma nova abordagem para medir o consumo energético de uma Rede Neuronal Profunda durante a fase de inferência. Esta métrica foi introduzida em funções de aptidão multiobjetivo de modo a conduzir a evolução no sentido de modelos energicamente mais eficientes. Para além disso, também introduzimos uma nova estratégia de mutação que permite a reutilização de módulos de camadas com probabilidade inversa ao seu uso de energia, (re)introduzindo assim conjuntos eficientes de camadas num modelo. Promomos também a introdução de uma camada de saída adicional conectada a uma camada intermediária de um modelo de Rede Neuronal Profunda e o posterior particionamento em dois modelos separados de modo a obter modelos de menor dimensão, porém de precisão semelhante e de consumo energético inferior. Até onde sabemos, nenhum trabalho prévio utiliza uma abordagem semelhante. Desenvolvemos uma estratégia que permite inicializar o Fast-DENSER com modelos previamente treinados de modo a iniciar o processo evolutivo com modelos precisos e evoluí-los de modo a serem mais eficientes em termos energéticos.

Os resultados obtidos pelas nossas propostas mostram que é possivel reduzir o consumo de energia de Redes Neuronais Artificiais sem comprometer o seu desempenho preditivo. Concretamente, podemos obter modelos que consomem menos 19.50 W (19.5%) do que um modelo de referência, tendo uma precisão mais elevada que este por 0.2%. O melhor modelo encontrado em relação à energia consome menos 29.18 W (29.2%), tendo uma diminuição mínima no desempenho (menos de 1%).

Palavras-Chave

Neuroevolução, Rede Neuronal Artificial, Rede Neuronal Profunda, Computação Evolucionária, Eficiência energética, Aprendizagem Computacional

Acknowledgements

I want to thank my supervisors, Professor Penousal Machado and Professor Nuno Lourenço, for their constant advice, assistance, and specialized knowledge over the past year, which have been of immeasurable value. Moreover, I want to express my appreciation for introducing me to the world of Artificial Intelligence in your classes. I have been motivated by your teachings to improve myself and challenge my boundaries continuously.

I am also deeply thankful to my family - my parents, sister, and nephew - for their unwavering care, support, and understanding. Their presence has been instrumental in making this journey a success. I would also like to acknowledge my late grandparents, who constantly encouraged and inspired me.

To Quantunna and all my friends there, I want you to know how truly grateful I am for the moments we have shared together. During these challenging times, taking breaks from work and enjoying our music together has been an experience I will cherish forever.

I am grateful to my friends, especially those closest to me, for their constant friendship, care, and encouragement to pursue my aspirations. I also want to thank those I have lost touch with. I am positive that all of you have contributed to my journey.

I extend my gratitude to the Calouste Gulbenkian Foundation for their financial support through the Gulbenkian Plus Scholarship from my first year at university to the year preceding this thesis. I genuinely appreciate their belief in my potential and contribution to my academic growth.

This page is intentionally left blank.

Contents

Acronyms

AI Artificial Intelligence. 3, 5

ANN Artificial Neural Network. xiii, 1, 2, 4, 13, 19, 74

CFG Context-free Grammar. 10

CNN Convolutional Neural Network. 14, 15, 18

DENSER Deep Evolutionary Network Structured Evolution. 18, 19, 23

DNN Deep Neural Network. xiii, 1, 2, 3, 4, 13, 14, 15, 16, 17, 18, 19, 23, 24, 26, 27, 35, 73

DSGE Dynamic Structured Grammatical Evolution. 18

EA Evolutionary Algorithm. 1, 6, 8, 9, 20

EC Evolutionary Computation. 4, 5

ES Evolution Strategy. 2, 8, 19

Fast-DENSER Fast Deep Evolutionary Network Structured Representation. xiii, xv, 2, 4, 19, 23, 24, 27, 28, 29, 32, 65, 71, 73, 74, 75

FLOPs floating point operations per second. 15, 20

GE Grammatical Evolution. 10, 18

GP Genetic Programming. 8, 9, 10

GPU Graphics Processing Unit. 14, 15, 16, 23, 32, 73, 75

IoT Internet of Things. 3

LEMONADE Lemarckian Evolutionary Algorithm for Multi-Objective Neural Architecture Design. 19, 20

ML Machine Learning. xiii, 1, 2, 11, 12, 13, 15, 24

MOL Multi-Output Learning. 16

MONAS Multi-Objective Neural Architecture Search. 18

MOO Multi-Objective Optimization. 17, 20

NAS Neural Architecture Search. 20

NE Neuroevolution. 1, 2, 4, 17, 19, 23, 32, 65, 68, 72, 74

NSGA-II Nondominated Sorting Genetic Algorithm II. 20

RNN Recurrent Neural Network. 18

SGE Structured Grammatical Evolution. 18

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

List of Algorithms

This page is intentionally left blank.

Chapter 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 CO_2 equivalent emissions just for training during 15 days [1]. 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 machine learning processes is becoming increasingly important.

Artificial Neural Networks (ANNs) are a type of ML model inspired by biological neural networks. 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. 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. They utilize mechanisms to evolve solutions over multiple generations, such as selection, crossover, and mutation. The process begins with a randomly initialized population whose evolution is steered by a fitness function that measures an individual's quality. 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

[2]. It can optimize the ANN's architecture and learning parameters.

One approach to address the energy inefficiency issue is 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 neural network models by using their accuracy as the fitness function, thus guiding the search towards accurate models [3].

The main motivation of our work is the inefficiency of Deep ML models regarding energy consumption. Over 90% of ML research tackles the accuracy metric alone [4]. Conversely, day-to-day usage requires balancing the model's accuracy and energy efficiency. On a larger scale, enabling efficient models could widen the use of Deep ML on personal, commercial, and industrial fronts, thus promoting algorithms that increase efficiency. Moreover, this could help mitigate the environmental impact of Deep ML, which has been increasingly brought to attention due to the high energy consumption of model training and inference, as well as minimizing time spent on training, designing, and inferring models, thus providing faster services to the end user.

In this work, we propose novel approaches integrated into Fast-DENSER, which aim at finding 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 if 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. Even though comprehensive testing is yet to take place, we developed a strategy that allows initializing Fast-DENSER with a previously trained model to start the evolution with already accurate models and evolve them towards power-efficient models.

To better visualize and interpret results from experiments performed in Fast-DENSER, we developed a visualization tool, NeuroView, that enables interactive data visualization. It allows for comparing the metrics of a single experiment, comparing two different experiments, and inspecting a single individual model.

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

The results of this work show that it is possible to have DNN models with substantial inferior power usage. Specifically, we can obtain models that consume less 19.50 W (19.5%) than a baseline model whilst having an accuracy 0.2% higher. The best model found regarding power consumes less 29.18 W (29.2%) whilst having a tiny decrease in performance (less than 1%).

1.1 Research Questions

The main objective of this work is to provide a way to make DNNs physically feasible on a small and large scale, making them more energy-efficient. This work aims to answer the question: **"Is it possible to create neural networks that consume energy in the same order of magnitude as the human?"**.

Some secondary research questions (RQ) are posed as follows to answer the main question:

RQ1 — Can we train a single network that can be partitioned into a smaller, accurate, and efficient energy-wise partition? To answer this, we will perform tests on models with two outputs that are subsequently partitioned into two separate DNN models. These have the advantage of requiring only the training of the composite model and, tentatively, being accurate while requiring less energy since they possess fewer layers. For example, the larger partitioned model could be used on devices with more extensive computational resources or in scenarios where a large number of inferences are performed on Artificial Intelligence (AI) models. At the same time, the smaller one could be employed on devices that do not, such as smartphones or Internet of Things (IoT) devices, which nowadays have AI models applied to them through previous training and testing.

RQ2 — To what degree does penalizing large energy consumption drive the search for efficient networks? To answer this, we will analyze how important it is to consider energy consumption in a neuroevolutionary process and at what point it matters most, i.e., **can we evolve accurate networks and only consider their energy consumptions after having a satisfactory accuracy value?**

1.2 Contributions

The key contributions of this work can be encapsulated as follows:

- Introduction of power consumption of a DNN model as a metric on Fast-DENSER.
- Development of multi-objective fitness functions on Fast-DENSER, which consider the power consumption of a model.
- Reutilization of modules of layers on Fast-DENSER, fulfilled through introducing two new mutation operators and adding an archive of modules and their respective power consumption.
- Model partitioning, a strategy that allows the development of two separate models on a single training, with one of them having an inferior number of layers.
- Using previously trained models as the basis on which Fast-DENSER begins its evolutionary process, thus starting with an accurate model and allowing it to evolve towards a more power-efficient model.
- NeuroView, a web app that allows interactive visualization and analysis of results from Fast-DENSER.

1.3 Structure

Chapter 2 presents the essential background for the reader to understand the concepts behind this work, such as Evolutionary Computation (EC), ANNs, and NE. Chapter 3 shows the various approaches we developed to tackle the problem. Chapter 4 comprises the performed experimental study: the experimental setup, the experiments performed, and a comparison and discussion of their results. Chapter 5 presents a visualization tool developed to understand better the results of NE experiments. At last, chapter 6 presents this work's conclusion and topics to approach in the future.

Chapter 2

Background

2.1 Evolutionary Computation

Evolutionary Computation (EC) is a branch of Artificial Intelligence (AI) inspired by the Darwinian Theory of Evolution [5]. This theory states that individuals who most effectively compete for the resources available in their environment are favoured by natural selection. The population members are different in behavior and physical traits, which determine their fitness. It also considers that minor, random variations occur during the evolutionary process, thus making it possible to create new traits.

Initial potential solutions are generated at the beginning of an EC process. Iteratively, weaker solutions are stochastically filtered out, and slight random modifications are introduced. Over time, solutions become increasingly improved, and the process culminates in near-optimal solutions.

One important trade-off in EC is balancing exploration and exploitation. Exploration refers to the systematic exploration of novel regions in the search space. In contrast, exploitation involves the examination of areas within the search space that are near previously visited points [6]. When focusing more on exploration, unproductive regions of the search space may be prone to be searched, wasting computational resources and time and diverging from an optimal solution. On the other hand, focusing on exploitation increases the susceptibility to becoming trapped in local optima, thus failing to explore regions that could contain better solutions.

2.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are meta-heuristic optimization algorithms based on the principles of Darwinian evolution [7, 8]. They work by exploring possible solutions in the search space in a stochastic manner to solve a problem better. An example of a simple EA can be analyzed in Algorithm 1.

Algorithm 1 Simple Evolutionary Algorithm

```
Require: $pop\_size > 0$, $generations > 0$, $0 \leq crossover\_prob \leq 1$,
  $0 \leq mutation\_prob \leq 1$
  $population \leftarrow initialize(pop\_size)$
  $generation \leftarrow 1$
  while $generation \leq generations$ do
      $new\_population \leftarrow \emptyset$
      $i \leftarrow 1$
      while $i \leq pop\_size$ do
          $individual \leftarrow select(population)$
          if $random() < crossover\_prob$ then
              $mate \leftarrow select(population)$
              $individual \leftarrow crossover(individual, mate)$
          end if
          if $random() < mutation\_prob$ then
              $individual \leftarrow mutate(individual)$
          end if
          $new\_population \leftarrow new\_population \cup individual$
          $i \leftarrow i + 1$
      end while
      $population \leftarrow new\_population$
      $generation \leftarrow generation + 1$
  end while
```

The EA begins with the random generation of an initial population. Each individual is represented by its genotype, which can take many forms, such as binary strings, real-valued vectors, or a permutation of a set of integers. The encoding from genotype to phenotype and vice-versa is problem-dependent.

An individual's effectiveness in solving a problem can be assessed through a fitness function adequate to the problem. It is crucial since it is the factor that quantifies a solution's quality by considering its qualities and defects. A well-designed fitness function can determine the degree of alignment between a solution and the intended objective(s).

Similarly to the natural evolution processes, EAs utilize mechanisms to allow diversity in the population. A broader search of the search space and the escape from local minima is encouraged through selection, crossover, and mutation mechanisms. The last two have an associated probability of being applied to control the level of exploration and exploitation. Furthermore, selection can be divided into parent selection and survivor selection.

Parent selection regards the selection of individuals in the current generation who will participate in the recombination step, thus generating offspring and passing their genes to the following generations. Parent selection is stochastic and predisposed to individuals with higher fitness values. To balance exploration and exploitation, lesser-fit individuals can often generate offspring. There are several types of selection operators for parent selection. The most common are tournament selection, roulette wheel selection, and stochastic universal sampling.

Tournament selection is a relatively simple operator where a sample of individuals is stochastically picked, and the one with the highest fitness is selected. In the roulette wheel operator, the selection is proportionate to an individual's fitness, i.e., individuals with higher fitness are more likely to be selected. One can visualize a wheel with sections proportionate to the individuals' fitness values. In the centre of it, an arm spins and stochastically falls in one of those sections. Stochastic universal sampling extends the roulette wheel operator by performing one spin with multiple equally-spaced arms instead of multiple spins with only one arm, thus providing an unbiased selection [7].

Variation operators form new individuals from existing ones, thus creating new candidate solutions.

Crossover is a variation operator that usually takes two individuals and combines information from their genotypes into one or two offspring. It is also possible to input more than two parents even though it has no biological counterpart. Figure 2.1 presents an example of a two-point crossover that produces two offspring. Two cutoff points are randomly selected to perform it, and the genotype material exchange is performed around them.

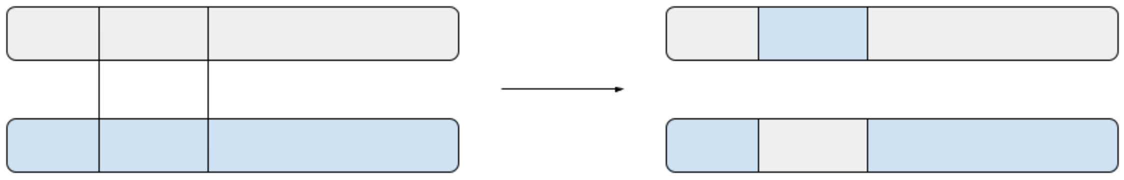


Figure 2.1: Example of two-point crossover

Mutation is a variation operator applied to a single individual that stochastically modifies its genotype. For a binary representation, the bit-wise mutation is usually used. In the case of permutation representation, there are many types of mutation operators, such as swap mutation. The first flips one or more bits of the genotype, and the latter randomly picks two genes and swaps their values.

Survivor selection is the selection of the individuals to be passed to the following generation. The used criterion can be fitness or age. In the first case, the fittest individuals are selected. In the second, the individuals have a maximum limit of generations to exist. In the case of age-based survivor selection, the fittest individuals are not necessarily selected since only their age is considered.

Multiple criteria determine when an evolutionary process terminates. For example, it may be when a known optimal fitness level has been reached, the maximum number of generations has been reached, the population diversity has fallen under a specific threshold, or the maximum allowed CPU time has been reached.

2.1.2 Evolution Strategies

Evolution Strategys (ESs) work similarly to EA, taking advantage of key concepts such as selection and mutation to aid the search for a solution to a problem [9, 10]. Considering $\mu \in \mathbb{N}$ as the number of parents and $\lambda \in \mathbb{N}$ as the number of offspring, ES algorithms are generally of the form (μ, λ) with $\mu < \lambda$ or $(\mu + \lambda)$ with $\mu \leq \lambda$.

In the case of the first form, the process begins with a population of λ randomly initialized individuals. The μ with the best fitness values are selected, producing (λ/μ) new individuals who replace the parents. The value of λ controls the size of the population. Thus, as it increases, the algorithm tends towards exploration. The value of μ controls the selective behaviour of the algorithm, with low values pushing it towards exploitative search. Since the only mechanism varying the individuals is mutation, if the mutation degree is too high, the children will be too different from the parents, thus probably resulting in individuals with inferior fitness.

In the case of the additive strategy, instead of replacing the parents, a new generation is comprised of μ parents plus λ new children. This way, only the first generation is of size λ. The following generations are of size $\lambda + \mu$.

2.1.3 Genetic Programming

Genetic Programming (GP) works similarly to EA. However, it concerns the evolution of programs, expressions, or models that can be represented, for example, as syntax trees [7]. To solve a GP problem, it is required to specify the terminal set (variables or constants) and the function set, as well as standard requirements to EA, such as the fitness function, the control parameters, and the termination criteria [11]. Considering a function set $F = \{+, -, *, /\}$ and a terminal set $T = \{\pi, r, 1, 2\}$, an example of a syntax tree is shown in Figure 2.2.

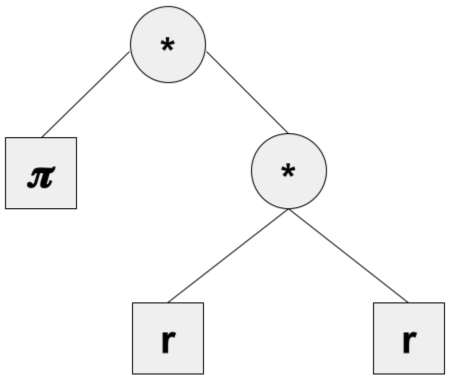


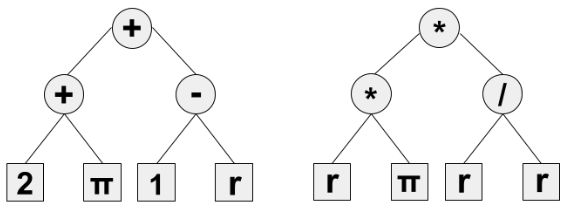
Figure 2.2: Example of a syntax tree

The initial population is created randomly, and its individuals are usually generated with the following methods: full, grow, and ramped half-and-half. The first method creates each tree branch with the maximum defined depth (Figure 2.3a). The second creates a tree where each branch may be as deep as the maximum defined depth (Figure 2.3b). The last method uses the previous two with equal probability (Figure 2.3c).

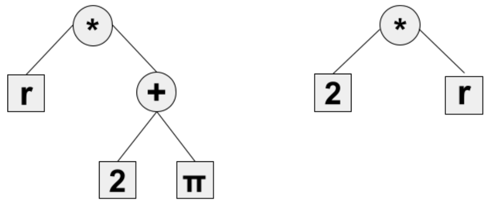
Similarly to EA, mutation and recombination operators exist. Mutation in GP can be accomplished by replacing the sub-tree with its root in a randomly selected node with a new randomly generated tree, as seen in Figure 2.4. On the other hand, recombination can be achieved through a sub-tree crossover. This crossover consists of swapping the sub-trees starting at two randomly selected nodes of two individuals, as shown in Figure 2.5.

While searching for solutions using GP, the solutions may become larger and more complex over time. If its fitness does not increase, it is called "bloat." Bloat can slow down the search and create solutions that are difficult to understand.

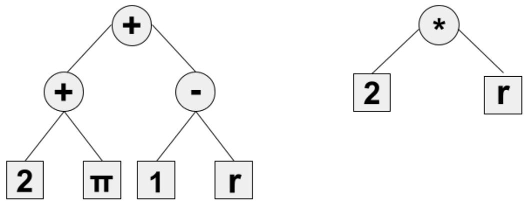
One way to prevent bloat is to use a fitness function that considers the tree's size and penalizes larger trees. Methods like double tournament, which first selects solutions based on their fitness and then on their size, can also help reduce bloat.



(a) Full method



(b) Grow method



(c) Ramped half-and-half method

Figure 2.3: Example of tree initialization methods for a population of two individuals

2.1.4 Grammatical Evolution

Grammar-based GP has a search space defined by a human-readable grammar [12]. Grammatical Evolution (GE) is a grammar-based GP with a clear distinction between genotype and phenotype. The first is a linear sequence of bits, and the latter is a high-level program. The mapping from genotype to phenotype is performed using the production rules of a Context-free Grammar (CFG) [13]. A grammar is represented by the tuple (N, T, P, S) with N being the set of non-terminals, T being the set of terminals, P being the set of production rules, and $S \in N$ is a start symbol.

The mapping from genotype to phenotype is performed as follows: the linear sequence of bits is read from left to right, and, starting with S, a production rule is iteratively applied to expand the leftmost non-terminal symbol. The value of the linear sequence being read (the codon) is used to select which production to apply by calculating the modulo of it per the number of existing choices for expanding the leftmost non-terminal symbol.

Consider the grammar in Figure 2.6 with N = $\{< expr >, < op >, < trig >, < var >\}$, T = $\{+, -, \sin, \cos, x, y, ()\}$ and S = <expr>. Considering a genotype with

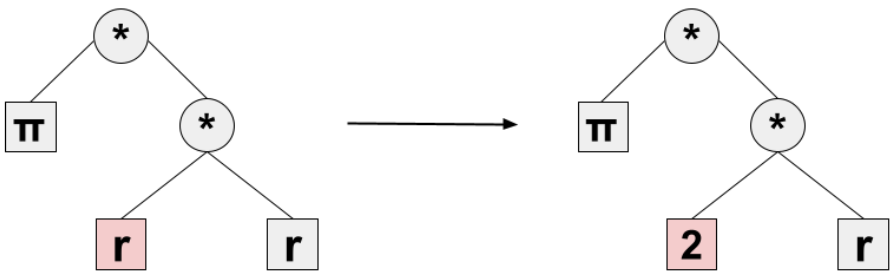


Figure 2.4: Example of sub-tree mutation

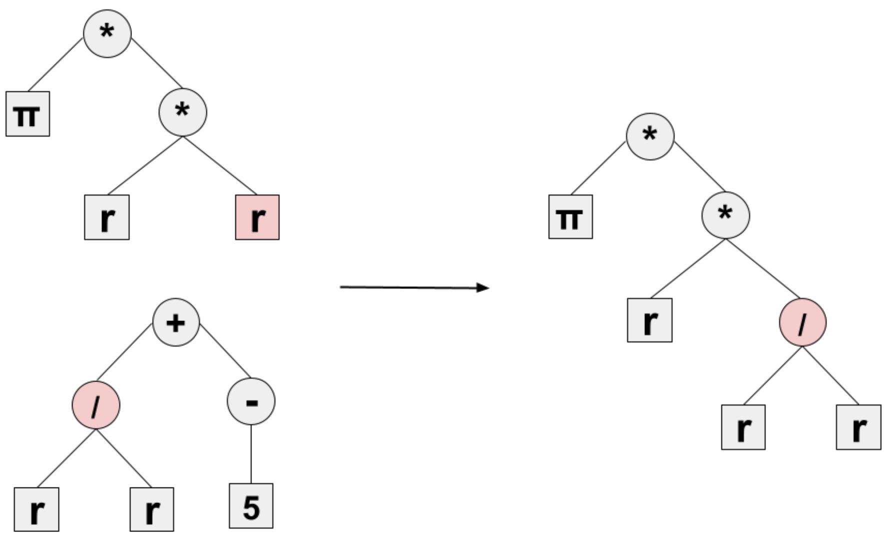


Figure 2.5: Example of sub-tree crossover

integer values ranging from 0 to 63, the mapping process is as follows (Figure 2.7). Firstly, we begin with the start symbol, <expr>, and read the first element of the integer vector. Since there are three options on how to expand the start symbol, we have $46 \mod 3 = 1$; thus, the start symbol is replaced by its second expansion rule, <trig>(<expr>). Following this, the second value in the genotype is read, and we expand the leftmost non-terminal symbol, i.e., <trig>. Since it has two expansion rules, $15 \mod 2 = 1$ and the second expansion rule is selected, replacing <trig> and forming cos(<expr>). The mapping process is executed until there are no more non-terminal symbols. If there are no more integers to read from the genotype, it is possible to use a wrapping mechanism instead of terminating the mapping process, thus reusing the genotype values.

2.2 Artificial Neural Networks

Machine Learning (ML) is an approach to data analysis that automates analytical model building. It is based on the idea that systems can learn from data, recognize

```
<expr>  ::= <expr> <op> <expr>
          | <trig>(<expr>)
          | <var>
<op>    ::= +
          | -
<trig>  ::= sin
          | cos
<var>   ::= x
          | y
```

Figure 2.6: Example of a grammar

Genotype: [46, 15, 17, 28, 50, 42, 22, 19, 51, 35]

<expr>	
<trig>(<expr>)	46 mod 3 = 1
cos(<expr>)	15 mod 2 = 1
cos(<var>)	17 mod 3 = 2
cos(x)	28 mod 2 = 0

Figure 2.7: Example of Grammatical Evolution mapping

patterns, and make decisions with minimal or no human intervention. In ML, a system is trained on a dataset, which enables it to learn from the data and perform predictions without being explicitly programmed. Several types of ML exist, such as supervised learning, unsupervised learning, and reinforcement learning [14].

In supervised learning, the model is trained on labelled data, where each input is associated with a corresponding output. The model learns to map inputs to the correct outputs by analyzing the labelled examples. The dataset is commonly split into training, validation, and testing. As their names imply, the training subset is utilized for training the model, the validation subset is employed to assess the model's performance and refine it during the training phase, and the final subset is reserved to evaluate the trained model's performance and measure its accuracy using unseen data.

Contrarily to supervised learning, unsupervised learning does not require labelled data since it learns from the inputs alone. Most unsupervised learning algorithms usually perform a clustering step and then learn to classify the patterns into a finite number of classes.

Reinforcement learning is similar to supervised learning but relies on the output's rightness instead of it being equal to an exact desired outcome, i.e., through the maximization of a reward function. This is achieved through a trial-and-error learning approach where an agent's actions in an environment impact its rewards and long-term rewards.

Artificial Neural Networks (ANNs) are a supervised ML model inspired by Biologic Neural Networks [15]. 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 (Figure 2.8). 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.

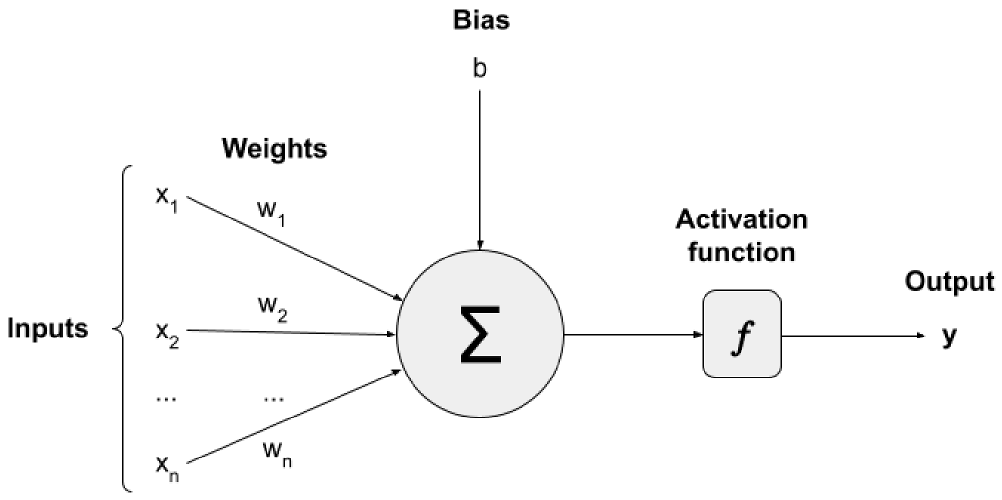


Figure 2.8: Example of an artificial neuron with *n* inputs

The perceptron is a single-layer ANN composed of one or more inputs and a single output, as depicted in Figure 2.9, with x as the input vector, w as the weight vector, f as the activation function, and y as the output [16]. During the training process, the weights of the input nodes are adjusted to improve the accuracy of the perceptron's classification of input data into one of two classes. As a linear classifier, it can only differentiate linearly separable data.

The multilayer perceptron extends the concept of perceptron by allowing more than one layer. It consists of an input layer, one or more hidden layers, and an output layer. Contrarily to the simple perceptron, it can handle non-linear classification tasks. Figure 2.10 shows a multilayer perceptron with three inputs, two hidden layers, and two outputs. The first hidden layer has three units, and the second one has four.

Deep Neural Networks (DNNs) are ANNs composed of many hidden layers. Due to this, DNNs can avoid the feature engineering step, which requires human

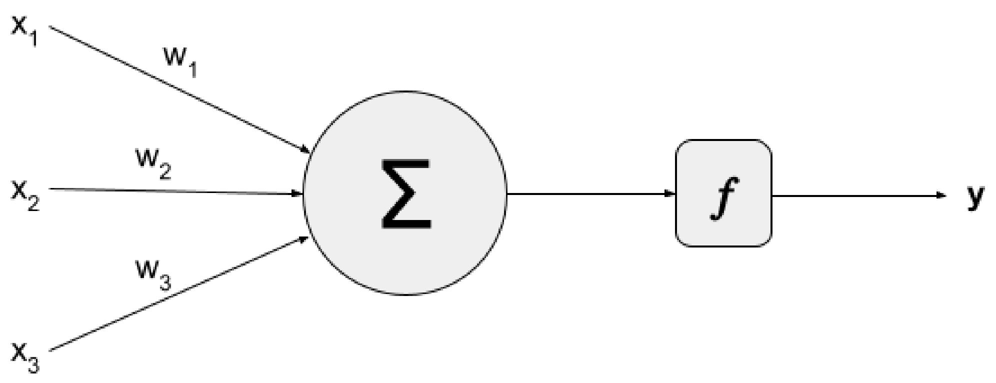


Figure 2.9: Example of a perceptron with three inputs

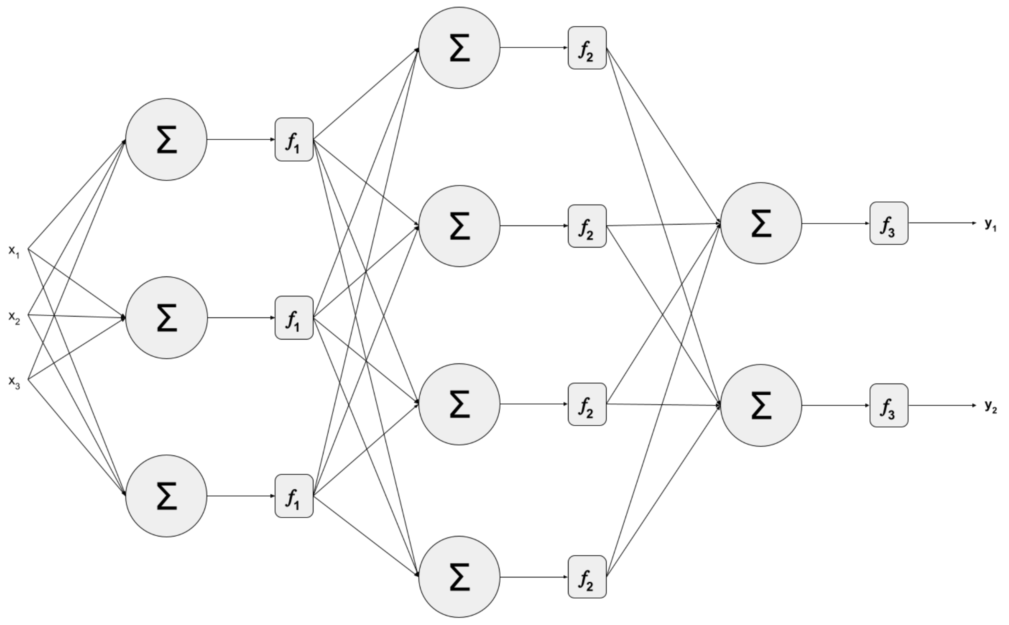


Figure 2.10: Example of a multilayer perceptron

expertise, by automatically discovering the representations needed for classification [17]. 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).

Convolutional Neural Networks (CNNs) are a type of DNN primarily used in image-driven pattern recognition problems due to their ability to capture spatial dependencies since, in an image, it is relevant to consider a pixel and its surrounding region [18]. CNNs have achieved remarkable success in computer vision tasks such as image classification, medical image analysis, and image segmentation. Their ability to automatically learn hierarchical representations and capture spatial dependencies has made them a fundamental tool in deep learning and computer vision.

At the basis of a CNN lies using one or more convolutional layers. These layers perform a convolution operation on the input data, which helps extract relevant features and learn hierarchical representations. The convolution operation involves sliding a small filter or kernel window over the input data. Each kernel extracts specific patterns or features from the input by performing a dot product between the sliding window and the filter. The resulting output is a feature map representing certain features at different spatial locations in the input.

2.2.1 Energy Consumption

With the increasing usage of computing resources and the growing issue of climate change, it is essential to produce energy-efficient hardware and software. GPU usage is typically energy-expensive due to arduous cycles such as training ML models, mining cryptocurrencies, and video games.

There are two main categories of measuring GPU power: through physical devices and through metrics that serve as a proxy for calculating the used power [19].

In the case of using physical devices, there are techniques such as using a power measuring tool or GPU drivers. Relatively to the first case, a device is placed between the power outlet and the computer's power cable. This way, it can output the measured power directly into a computer. Relatively to the other option, it is possible to use a program to read the GPU power measuring device if supported. In the case of Nvidia, it is possible to use *nvidia-smi* or third-party programs such as *pyJoules* [20], which allows obtaining energy consumption programmatically in Python.

As for the case of using metrics to estimate energy consumption, it is possible to use the number of trainable parameters in a model or the number of floating point operations per second (FLOPs). While both are good indicators, literature shows that FLOPs provide a better proxy for energy consumption [21].

It is also possible to estimate the energy consumption of a DNN through frameworks such as *NeuralPower* [22]. It is a layer-wise predictive framework based on sparse polynomial regression that allows the estimation of the serving energy consumption of a network deployed on any GPU, i.e., it is not restricted to the manufacturer.

[23] presents a thorough comparison of power and energy efficiency of various well-known DNNs such as AlexNet and GoogleNet through the convnet benchmark suite [24]. Training frameworks like Caffe [25], Torch [26], and Tensorflow [27] are compared as well. It is concluded that Torch is the most efficient framework on GPU and that Caffe has a similar performance, depending on the GPU being used. It also shows that convolutional layers consume most of the total energy consumption, followed by fully-connected layers. It is also noteworthy that increasing the batch size, i.e., the number of examples used in one iteration, increases power consumption but reduces the energy consumption per image.

It is possible to develop problem-specific networks comprising a few neurons [28], taking inspiration from simple biological organisms. Since the outputs of this kind of network are as accurate or even more accurate than the larger counterparts, they present benefits since a lower number of neurons results in lower energy consumption. Similarly, it is sometimes preferable to separate feature extraction from decision-making to allow problems to be solved with more straightforward methods while maintaining comparable performance [29].

2.2.2 Multi-Output Learning

Multi-Output Learning (MOL) allows the prediction of multiple outputs with a single input, making it possible to solve more complex decision-making problems with multiple related solutions [30]. MOL makes it possible to predict n aspects from a single input without developing n different models, each specialized in one aspect. Instead, one model can have n output layers, each capable of predicting one aspect.

When having multiple output layers, each output layer requires an adequate loss function for the data type it is expected to predict. They are then combined into a single scalar value used in the optimization process. Specifying loss weights to apply to each loss value in the combined loss is possible.

For example, it is possible to predict the age, gender, and race of an individual with a three-output model, with the age output meant for numeric values and the other two for categorical values [31].

2.3 Neuroevolution

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 [2]. 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.

Multi-Objective Optimization (MOO) is the process of finding a solution or set of solutions that optimize multiple objective functions [32]. It is necessary to perform trade-offs between objectives in multiple problems among many fields, such as economics, engineering, and computer science. For example, the industrialized development of a product might require maximizing profit while minimizing costs and environmental impact.

A key concept of MOO is that of the Pareto optimal solution. It represents a solution that cannot be improved in any single objective without worsening the others. A set of these solutions comprises the Pareto frontier and represents all the solutions that are trade-offs between the objectives.

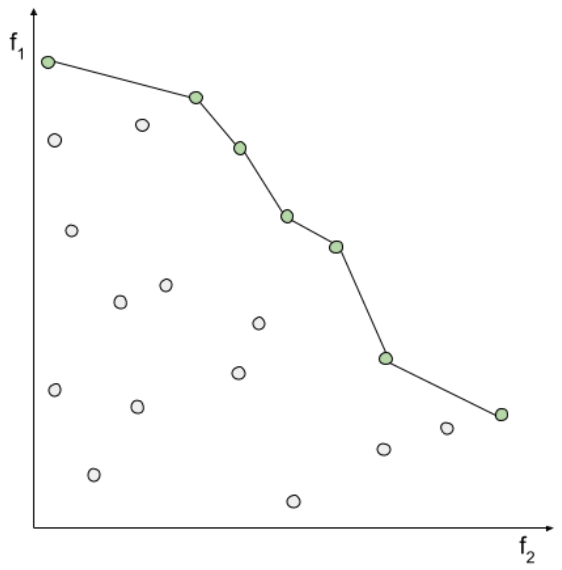


Figure 2.11: Example of the Pareto front of a set of solutions (maximizing f_1 and f_2)

2.3.1 MONAS

Multi-Objective Neural Architecture Search (MONAS) is a framework that employs reward functions that consider accuracy and other objectives, such as power consumption, when searching for DNN architectures [34].

MONAS is a two-stage framework with a generation and evaluation stage. In the former, a Recurrent Neural Network (RNN) is used as a robot network and generates a hyperparameter sequence for a CNN. In the latter stage, an existing CNN model is trained as a target network using the hyperparameters from the previous stage. The target network's accuracy and energy consumption are used as rewards to the robot network, which updates itself based on this reward with reinforcement learning.

The performed experiments use models of the AlexNet [35] and CondenseNet [36] families in the evaluation stage. The used dataset is CIFAR-10 [37], which has 50 thousand examples for training and 10 thousand for testing, where each example is a 32x32 color image, and the images are divided into ten classes.

Compared to the CondenseNet baselines, MONAS obtained more accurate models that consume less energy. The best model found is 0.14% more accurate than the best CondenseNet model tested whilst consuming 40.4% less energy than it. The most power-efficient model found consumes 13.6% less energy whilst being 0.62% more accurate than the worst-performing CondenseNet model tested.

2.3.2 DENSER

Deep Evolutionary Network Structured Evolution (DENSER), as the name suggests, is an algorithm that allows the search of DNNs through a grammar-based neuroevolutionary approach that searches both network topology and hyperparameters [33].

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. Allied with dynamic production rules, DSGE allows creating multiple-layer DNNs. SGE proves to perform better than GE, and DSGE proves to be superior to SGE. 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 parameters.

Fast Deep Evolutionary Network Structured Representation (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 [3]. 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, DENSER obtained models with an accuracy higher than most of the state-of-the-art results, and on the CIFAR-100, 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.

2.3.3 LEMONADE

Lemarckian Evolutionary Algorithm for Multi-Objective Neural Architecture Design (LEMONADE) is an evolutionary algorithm for multi-objective architecture search that allows approximating the entire Pareto front of architectures under multiple objectives, such as predictive performance and number of parameters in a single run of the method [38].

It uses a Lamarckian inheritance mechanism, which generates child networks that are started with the predictive performance of their trained parents. It is accomplished by using approximate network morphism operators for generating children. These operators preserve the network's function, thus not requiring the training from scratch and reducing the required training time per network.

LEMONADE keeps a population of networks on an approximation of the Pareto front of the multiple objectives. It exploits the fact that evaluating particular objectives, such as the number of parameters on an architecture, does not require training the model, while evaluating the predictive performance on validation data does. That is, LEMONADE assigns higher probabilities to architectures with a better chance of providing more knowledge on the Pareto front for the "cheap" objectives and then evaluates only this subset, reducing the required computational resources. LEMONADE returns a set of architectures, thus not requiring the definition of weighted reward functions and allowing the user to select the most adequate.

The experiment performed on the CIFAR-10 dataset initializes LEMONADE with trivial architectures instead of state-of-the-art ones. It aims at minimizing five objectives: performance on CIFAR-10 and CIFAR-100 datasets, number of parameters, number of multiply-add operations, and the inference time. Relatively to the CIFAR-10 dataset and in comparison with literature results LEMONADE achieves an error rate 0.05% inferior to DPP-Net whilst using the same number of parameters, and achieves an error rate 0.28% superior to PLNT whilst using 8.39% less parameters.

2.3.4 NSGA-Net

NSGA-Net is an evolutionary approach for Neural Architecture Search (NAS) that utilizes at its core Nondominated Sorting Genetic Algorithm II (NSGA-II) as the MOO algorithm. It is a population-based search algorithm that explores a space of potential neural network architectures in three steps. At first, the population is initialized, taking into account knowledge from hand-crafted architectures. Afterward, an exploration step is performed through crossover and mutation of architectures. At last, an exploitation step uses a Bayesian network that has knowledge of the history of evaluated neural architectures [39].

This approach ponders two objectives useful for real-world deployment: maximizing accuracy and minimizing power consumption using FLOPs as a proxy.

The performed experiments use the CIFAR-10 dataset. When compared to the presented state-of-the-art architectures designed by human experts, NSGA-Net achieves a test error 0.38% inferior whilst using only 12.9% of the number of parameters. When compared with other NAS methods, it achieves similar accuracy and uses the same number of parameters as the best one. Furthermore, NSGA-Net obtained a model with the best test error (0.15% less than the previous best) although it uses 8.1 times the number of parameters.

Nondominated Sorting Genetic Algorithm II

NSGA-II combines MOO and EA by using evolutionary mechanisms to find solutions to an optimization problem [40, 41].

The algorithm starts by generating a population of potential solutions to the optimization problem and evaluating their performance in the multiple objectives. Following this evaluation, a fast non-dominated sorting is performed where the solutions in the population are sorted into different fronts based on their level of non-dominance. The first front consists of the non-dominated solutions. The second consists of the solutions dominated by at least one solution of the first front, and so forth. A crowding distance is assigned to solutions belonging to the same front to ensure diversity in the population. Selection is performed by taking the crowding distance measure as key to balancing exploration and exploitation. The selected solutions are then recombined and mutated, replacing the worst-performing solutions to keep a constant population size. This process is repeated until a satisfactory set of solutions is found.

This page is intentionally left blank.

Chapter 3

Approach

In this chapter, we introduce the modifications we propose to apply to Fast Deep Evolutionary Network Structured Representation (Fast-DENSER) to make it able to generate power-efficient Deep Neural Network (DNN) models.

3.1 Power Measurement

Measuring the power a Graphics Processing Unit (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. As described in Section 2.3.2, Deep Evolutionary Network Structured Evolution (DENSER) is a Neuroevolution (NE) framework and, as such, consumes energy on the search for optimal models. That 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 Machine Learning (ML) computations are inference processing [42] and about 60% of Google's ML energy usage is for inference with the remaining portion being for training [1].

With that in mind, 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.

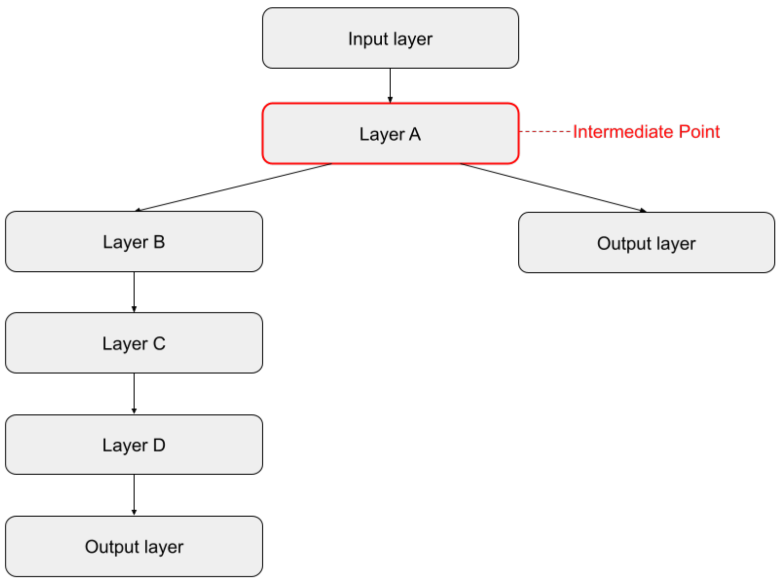
3.2 Model Partitioning

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.

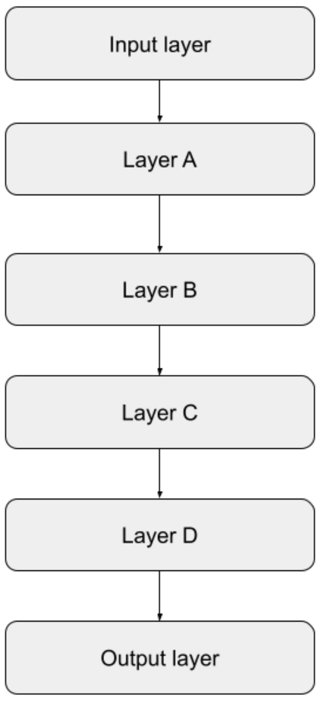
For that matter, we experimented with a modification to Fast-DENSER on which an extra output layer is connected to an intermediate layer of the model. The two-output model (Figure 3.1a) is trained to optimize for two outputs. At the validation step, it is split into left (Figure 3.1b) and right (Figure 3.1c) 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.

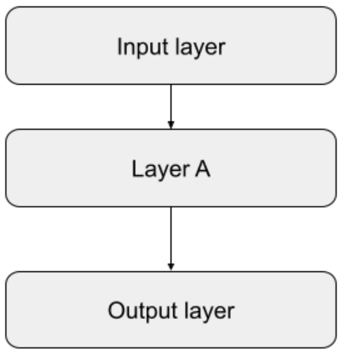
Since the maximum value of the point depends on the number of layers of the model, the grammar initializer and the mutation mechanism for the macro level were modified to consider the maximum number of layers of the model dynamically. To introduce the intermediate point in the evolutionary process, it was considered part of the macrostructure and, as such, as a rule of the grammar. The introduced rule is *<middle_point> ::= [middle_point,int,1,0,x]*, which means that one



(a) Full model



(b) Left partition



(c) Right partition

Figure 3.1: Example of a two-output model and its left and right partitions, with the layer marked by the intermediate point in red color.

integer value is obtained with the lower limit being zero. The upper limit is an arbitrary variable x that will be replaced at any instance by the maximum number of layers in the model.

3.3 Multi-objective 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., $power^{-1}$.

In light of our hypothesis suggesting 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. This aims to enhance the accuracy of both partitions. 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 3.1, 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. 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 steer the evolution properly.

$$f_1 = \min(0.80, acc_{left}) + \min(0.85, acc_{right}) + power_{left}^{-1} \tag{3.1}$$

With this in mind, another fitness function was designed (Equation 3.2), which multiplies the power usage by 10, thus giving it a more considerable weight and bringing it to another order of magnitude. Optimizing this weight value is out of the scope of this work since it could be a research task in itself. 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, as expected. Since this is not an optimal behavior, a function that does not limit accuracy was developed.

$$f_2 = \min(0.80, acc_{left}) + \min(0.85, acc_{right}) + 10 * power_{left}^{-1} \tag{3.2}$$

As shown in Equation 3.3, the mentioned fitness function considers only the partitions' accuracy values when both are inferior to a threshold. After any of the partitions surpasses its designated threshold, power consumption is added to the fitness function 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 obtain accurate and efficient models.

$$f_3 = \begin{cases} acc_{left} + acc_{right} & \text{if } acc_{left} \leq 0.80 \land acc_{right} \leq 0.85 \\ acc_{left} + acc_{right} + 10 * power_{left}^{-1}, & \text{otherwise} \end{cases} \tag{3.3}$$

3.4 Module Reutilization

Internally, Fast-DENSER considers modules of layers on each individual from which a DNN is then unraveled. 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 3.3), inaccurate models are intensely penalized and, as such, discarded in favor of better ones.

Whenever a module of layers is randomly generated or modified, its power consumption is measured. To do this, a temporary model 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 to simplify the process. The measured power consumption is then stored in a hash map where the module is the key, and the power consumption is the value.

An operator of mutation, *reuse module*, was introduced to use this information. It selects a module with a probability inversely proportionally to its power consumption, i.e., modules with inferior power consumption have a superior probability of being chosen. As shown in Equation 3.4, 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.

$$P(i) = \frac{\frac{1}{power_i}}{\sum_{j=0}^{n} \frac{1}{power_j}} \tag{3.4}$$

We also introduced an operator that randomly removes a module from an individual to counteract the operator reusing modules.

3.5 Seed Model

By default, Fast-DENSER begins the evolutionary process with an individual initialized from scratch and mutates it to create new individuals. Afterward, all individuals are trained and evaluated, with the best selected for the following generation.

Starting from scratch, the individual will most likely result in an inaccurate model due to its random inception. The new strategy we introduced in Fast-DENSER provides the framework with an already trained model accompanied by its weights and phenotype. By adopting this method, we can evolve existing models focusing on power efficiency, thus generating more efficient models. One limitation of this approach is requiring the phenotype that originated the model. That is, it is necessary for the model to have had origin in Fast-DENSER or to elaborate the phenotype manually.

Fast-DENSER initializes the first individual with the provided phenotype and generates variations of it as usual, i.e., as if the individual had been initialized from scratch. The provided phenotype is parsed, and the modules are initialized from their corresponding phenotype portion instead of undergoing random generation.

The conversion from phenotype to genotype is crucial in this process, and an algorithm has been developed to do so. It translates a phenotype, essentially a list of properties, into a structured genotype representation as required by Fast-

DENSER. The algorithm searches for the symbols and matching rules in the grammar to establish connections between phenotype properties and their corresponding genotype structures. This step enables Fast-DENSER to evolve the individual as if it had been generated by it.

Comprehensive testing of this approach has not yet taken place.

This page is intentionally left blank.

Chapter 4

Experimental Study

This chapter will present the experimental setup, performed experiments, comparison between them, and discussion of results.

4.1 Dataset

The dataset used in all experiments is Fashion-MNIST [43]. It is 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 as shown in Figure 4.1.

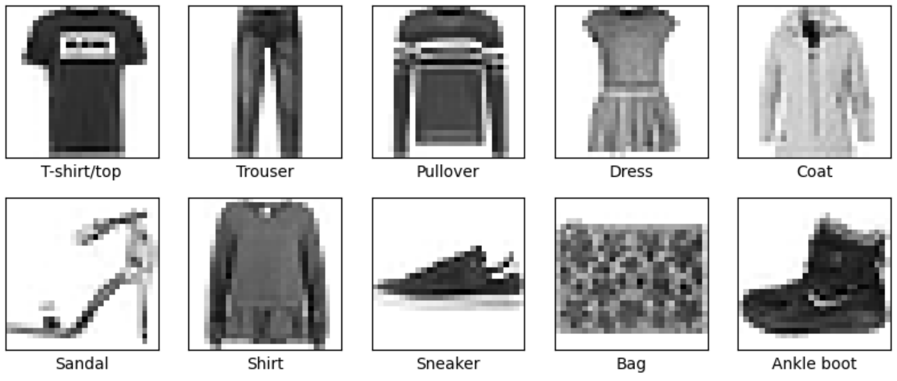


Figure 4.1: First instance for each class in the Fashion-MNIST training set.

Fashion-MNIST was developed as a more challenging replacement for the well-known MNIST [44] dataset by swapping handwritten digits with images of clothes such as shirts and coats, aiming at a more realistic and relevant benchmark.

Both the training and testing sets exhibit balanced data, with the former consisting of 6000 instances for each class and the latter containing 1000 instances for each class.

Fashion-MNIST authors performed experiments with various classifiers with test accuracy ranging from 51.1% to 89.7%. These values have since been surpassed with the current best accuracy reported by [45] with 99.06%, followed by [46] with an accuracy of 96.91%

4.2 Experimental Setup

All 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.

Since power usage is essential in making NE physically plausible, a function to measure power with the *pyJoules* library was developed. Its pseudocode can be analyzed in Algorithm 2, with *meter* the library tool that facilitates the measurement of energy consumed, and with functions *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 milliJoule 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.

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.

Table 4.1 presents the experimental parameters used across the experiments. Except for the experiment with skip connections, all experiments considered null probabilities on *Add connection* and *Remove connection*. All the experimental analyses in this chapter consider the mean of the generation's best individual according to fitness over 5 runs.

Since some types of layers perform more operations and, as such, are prone to consume more energy, we will analyze in each experiment the layer composition of the models with emphasis on the distributions of every 30 generations. The

Algorithm 2 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 $(output, mean\_power)$
```

types of Keras layers considered in all the experiments are the following:

- Dense (*fc*) - fully connected layer where each neuron is connected to every neuron in the previous layer.
- Dropout (*dropout*) - prevents overfitting by setting part of the input units to zero.
- MaxPooling2D (*pool-max*) - downsamples the input by computing the maximum value of a window.
- AveragePooling2D (*pool-avg*) - downsamples the input by computing the maximum value of a window.
- BatchNormalization (*batch-norm*) - normalizes input activations for each batch
- Conv2D (*conv*) - applies two-dimensional convolution.

It should be noted that all line charts show the mean value of the data they represent, with the shadow serving to delineate a 95% confidence interval.

4.2.1 Statistical Tests

Throughout this chapter, we will conduct several statistical tests. We considered a significance level of $\alpha = 0.05$ across them.

We conduct Shapiro-Wilk tests whenever required to assess the normality of the data. In this test, the null hypothesis (H_0) is that the data follows a normal distribution, and the alternative hypothesis (H_1) is that the data does not follow a

Table 4.1: 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%
Add connection	0%, 15%
Remove connection	0%, 15%
DSGE-level rate	15%
Macro layer rate	30%
Train longer rate	20%

Train Parameter	Value
Default train time	10 min
Loss	Categorical Cross-entropy

normal distribution. If the p-value is less than α, we can reject the null hypothesis, thus proving the data does not follow a normal distribution.

To determine the correlation between two variables, we obtain Spearman's rank correlation coefficient. This is favored relative to the Pearson correlation coefficient in the cases where the data does not follow a normal distribution. In Spearman's rank, the null hypothesis (H_0) is that there is no monotonic correlation between the two variables, and the alternative hypothesis (H_1) is that there is a monotonic correlation between the two variables.

We perform the non-parametric Kruskal-Wallis test when comparing more than two groups that do not follow a normal distribution and with independent samples. In this test, the null hypothesis (H_0) is that the median is equal across all groups, and the alternative hypothesis (H_1) is that the median of at least one group is different from the others.

In the cases where we verify significant differences between the groups through the Kruskal-Wallis test, we perform the Mann-Whitney U test post-hoc between the groups and adjust the p-values using Bonferroni correction to offset the multiple comparisons problem. The Mann-Whitney U test has the following hypothesis: the null hypothesis (H_0) is that the two populations are equal, and the alternative hypothesis (H_1) is that the two populations are unequal.

4.3 Baseline Experiment

We present a baseline experiment conducted to assess whether the energy consumption of a DNN increases when only accuracy is considered in the fitness function.

Correlation between accuracy and power usage

Figure 4.2 presents the accuracy and power usage evolution of each generation's best individual over 150 generations. Accuracy increases faster at the beginning of the evolutionary process, accompanied by increased power usage. After generation 50, it is noticeable that the power usage decreases significantly but is soon followed by a significant increase. Over the remaining execution, accuracy slowly increases, and the power usage stabilizes.

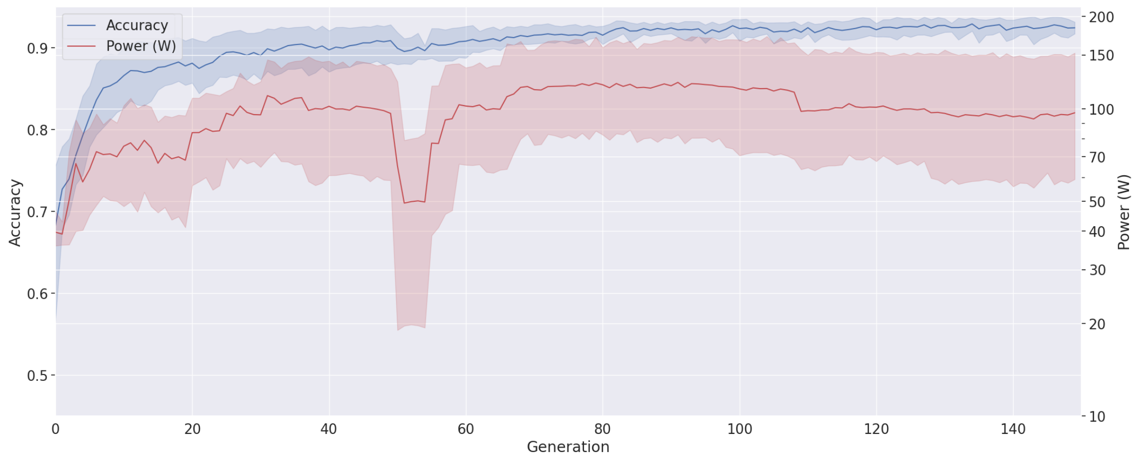
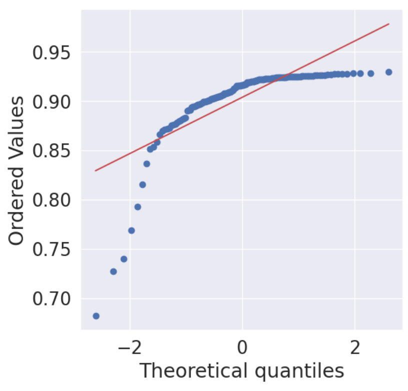


Figure 4.2: Evolution of accuracy and power usage over 150 generations of the baseline experiment.

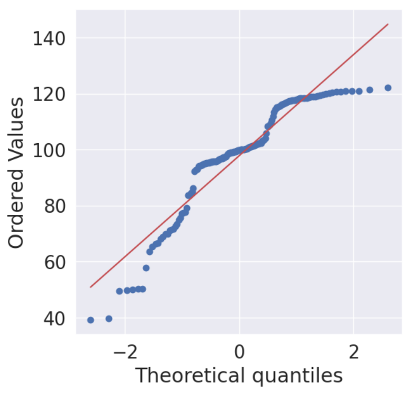
To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on accuracy yielded a value of 0.601 with a corresponding p-value of 1.64×10^{-18}, and on power, it resulted in a value of 0.893 with a p-value of 5.77×10^{-9}. Since, in both cases, the p-value is less than α, we reject the null hypothesis.

Following the assessment of normality using both Q-Q plots (Figures 4.3a and 4.3b) and the Shapiro-Wilk tests, it was determined that the data does not follow a normal distribution.

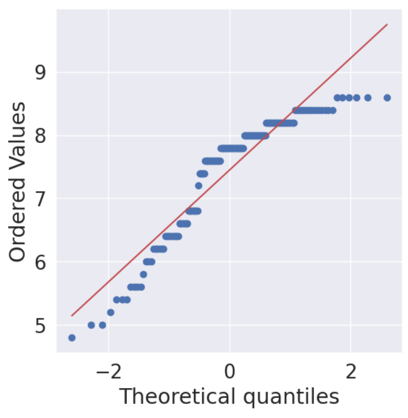
To formally assess the correlation between accuracy and power usage, we calculated Spearman's rank correlation coefficient, which showed $\rho = 0.417$ and a p-value of 1.12×10^{-7} which is less than α. This indicates that the correlation be-



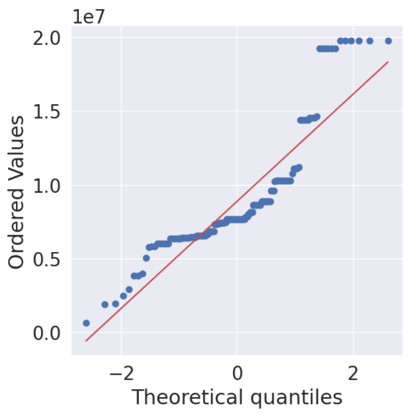
(a) Accuracy



(b) Power usage



(c) Num. of Layers



(d) Trainable Parameters

Figure 4.3: Q-Q plots showcasing the distribution comparisons for accuracy, power usage, number of layers, and number of trainable parameters over five runs of the baseline experiment.

tween accuracy and power usage is statistically significant, thus suggesting that there is a trend in how power usage increases with accuracy.

Correlation between the number of layers and power usage

Figure 4.4 presents the number of layers and power usage evolution of each generation's best individual over 150 generations. Generally, it is noticeable that the line relative to the power usage follows the one relative to the number of layers with the observable exception of generations 50 through 60, where the power usage drops without an apparent decrease in the number of layers. This unusual decrease might be due to measurement errors.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on the number of layers resulted in a value of 0.877 with a corresponding p-value of 8.10×10^{-10}. We reject the null hypothesis since the p-value is less than α.

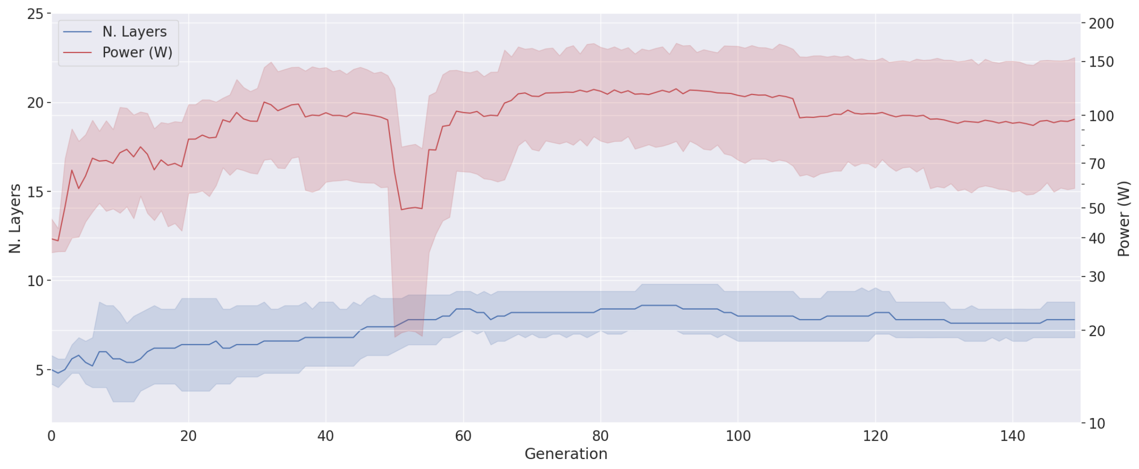


Figure 4.4: Evolution of the number of layers and power usage over 150 generations of the baseline experiment.

Following the assessment of normality using both Q-Q plots (Figures 4.3b and 4.3c) and the Shapiro-Wilk tests, it was determined that the data does not follow a normal distribution.

To formally assess the correlation between the number of layers and power usage, we calculated the Spearman's rank correlation coefficient, which showed $\rho = 0.778$ and a p-value of 1.05×10^{-31} which is less than α. This indicates that the correlation between the number of layers and power usage is statistically significant, thus suggesting that there is a trend in how power usage increases with the number of layers.

Correlation between the number of trainable parameters and power usage

Figure 4.5 shows how the number of trainable parameters and power usage evolve over 150 generations. We can observe an apparent trend when, in the beginning, the power usage increases as the number of trainable parameters increases. When around generation 30, the number of parameters starts to decrease, the power usage does not follow such decrease. Except for generations 50 through 60, the power usage stabilizes while the number of trainable parameters stabilizes.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on the number of trainable parameters resulted in a value of 0.819 with a corresponding p-value of 2.46×10^{-12}. We reject the null hypothesis since the p-value is less than α.

Following the assessment of normality using both Q-Q plots (Figures 4.3b and 4.3d) and the Shapiro-Wilk tests, it was determined that the data does not follow

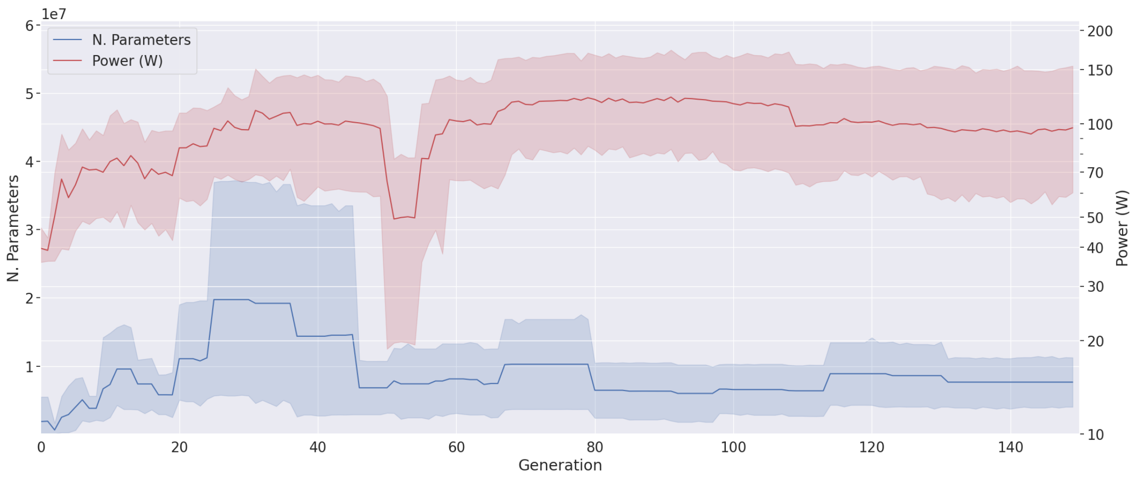


Figure 4.5: Evolution of the number of trainable parameters and power usage over 150 generations of the baseline experiment.

a normal distribution.

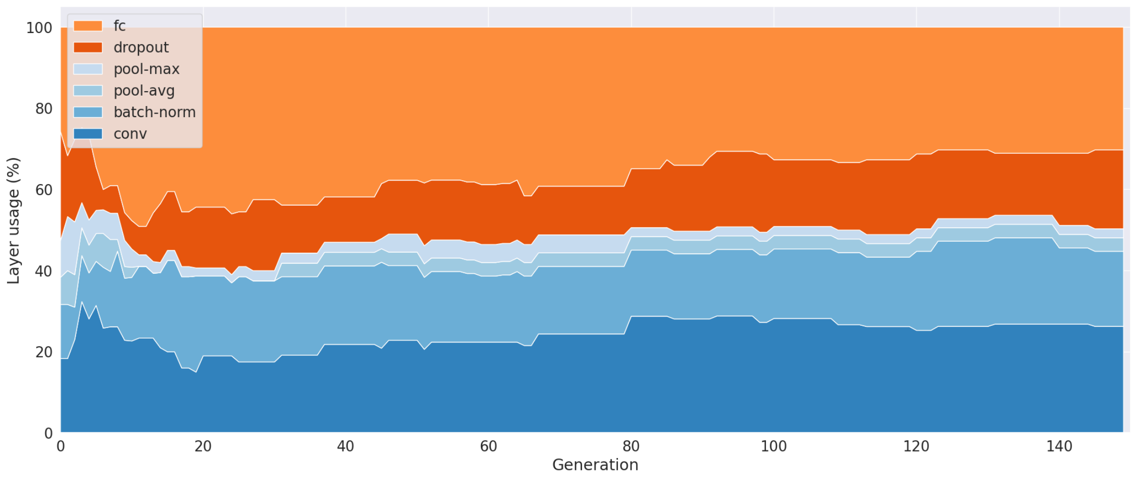
To formally assess the correlation between the number of trainable parameters and power usage, we calculated Spearman's rank correlation coefficient, which showed $\rho = 0.072$ and a p-value of 0.383, greater than α. This indicates that the correlation between the number of trainable parameters and power usage is not statistically significant. This lack of correlation might be explained by the fact that what matters most regarding power consumption is not the number of parameters but the number of operations performed by the layers (as mentioned in Section 2.2.1).

Layer composition breakdown

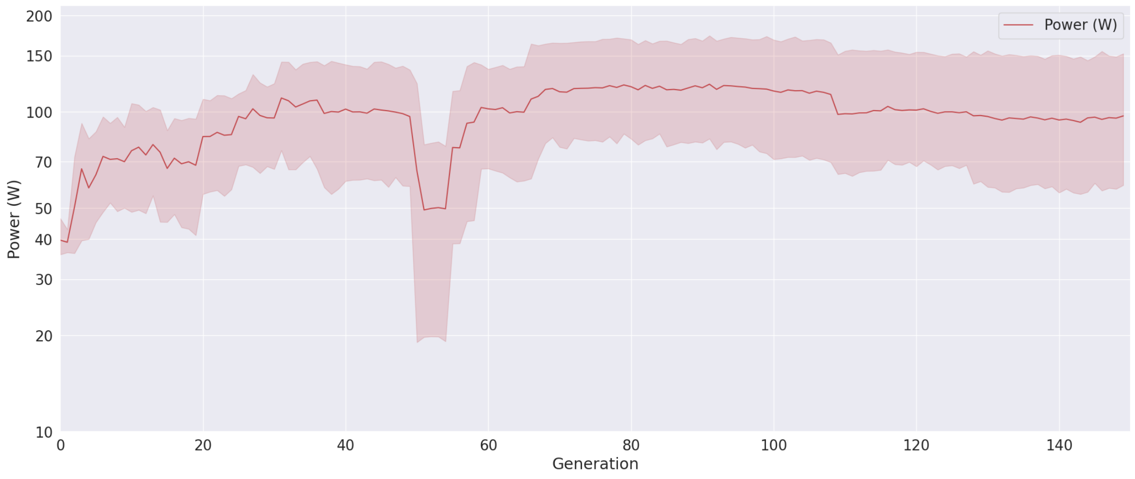
The evolution of layer composition of each generation's best model by percentage is presented in Figure 4.6a with Figure 4.6b presenting the power usage over 150 generations. Table 4.2 showcases the values of every 30th generation to facilitate the analysis.

As can be observed in Figure 4.6a, there's a noticeable uptrend in power usage on the initial 40 generations, with *fc* layers dominating and increasing in prevalence. Although *dropout* layers initially decline lightly, they rebound after about 90 generations. *conv* layers show steady growth, rising from 17.5% on the 30th generation to 26.3% on the last generation. After around the 100th generation, the layer composition seems to to have stabilized and, at the same time, power usage slightly decreases, showing that this effect is likely due to some other factor. The last generation culminates on models with a predominance of *fc* layers, followed by *conv*.

Comparing the first and last generations, we observe that the latter has a higher power consumption and an increased percentage of *fc*, *batch-norm*, and *conv* layers than the former, which shows us that these types of layers are prone to consume more power. Contrarily, the last generation presents less *dropout* and pooling layers, which ought to consume less power.



(a) Layer composition



(b) Power usage

Figure 4.6: Evolution of layer composition by percentage and power consumption over 150 generations of the baseline experiment.

Summary

We proved there is a positive correlation between accuracy and power usage and between the number of layers and power usage. We also proved that the correlation between the number of trainable parameters and power usage is not statisti-

Table 4.2: Evolution of layer composition by percentage over 150 generations of the baseline experiment, showcasing data for every 30th generation.

Layer	**Generation**					
	1	**30**	**60**	**90**	**120**	**150**
fc	25.7% ± 14.0%	42.5% ± 16.6%	38.8% ± 28.7%	34.0% ± 12.3%	32.7% ± 11.9%	30.2% ± 9.9%
dropout	27.0% ± 22.2%	17.5% ± 6.1%	14.8% ± 10.1%	16.3% ± 5.1%	18.4% ± 12.1%	19.4% ± 12.4%
pool-max	9.0% ± 12.4%	2.5% ± 5.6%	4.4% ± 9.9%	2.2% ± 5.0%	2.2% ± 5.0%	2.2% ± 5.0%
pool-avg	6.7% ± 14.9%	0.0% ± 0.0%	3.3% ± 7.5%	3.3% ± 7.5%	3.3% ± 7.5%	3.3% ± 7.5%
batch-norm	13.3% ± 29.8%	20.0% ± 11.7%	16.3% ± 8.0%	16.1% ± 8.2%	17.1% ± 7.7%	18.5% ± 8.0%
conv	18.3% ± 20.7%	17.5% ± 11.7%	22.4% ± 16.1%	28.1% ± 4.4%	26.2% ± 6.6%	26.3% ± 4.7%

cally significant. From this, we can conclude that the number of layers increases the power consumption of a model, whilst the number of trainable parameters does not.

The results of this baseline experiment show that the evolution tends towards models with increased complexity to achieve higher accuracy values without considering power efficiency, thus setting the motivation for our work.

4.4 Power Experiment

This experiment utilizes the approaches described in Sections 3.1 (measuring power usage), 3.2 (model partitioning), 3.3 (multi-objective fitness functions), and 3.4 (module reutilization). The employed fitness function is presented in Equation 3.3.

Correlation between accuracy and power usage

Figure 4.7 presents the accuracy and power usage evolution of each generation's best individual over 150 generations. Accuracy increases sharply in the beginning of the evolutionary process, stabilizing afterwards. Power usage increases at the beginning but diminishes around the 30th generation, having a somewhat substantial increase around the 90th generation, stabilizing afterwards. The decrease is most likely because the fitness function only starts penalizing power consumption after reaching a certain accuracy level.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on accuracy yielded a value of 0.565 with a corresponding p-value of

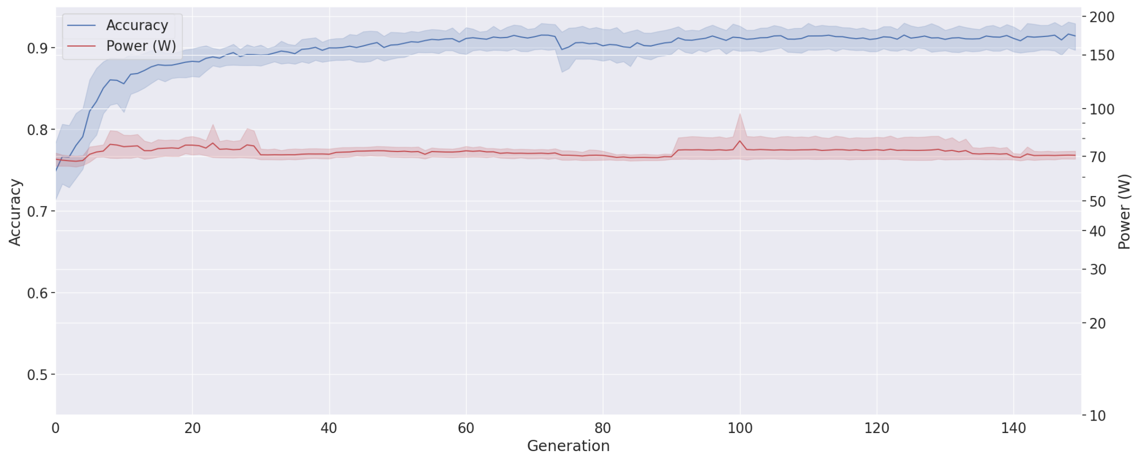


Figure 4.7: Evolution of accuracy and power usage over 150 generations of the power experiment.

2.86×10^{-19}, and on power, it resulted in a value of 0.976 with a p-value of 1.11×10^{-2}. Since, in both cases, the p-value is less than α, we reject the null hypothesis.

Following the assessment of normality using both Q-Q plots (Figures 4.8a and 4.8b) and the Shapiro-Wilk tests, it was determined that the data does not follow a normal distribution.

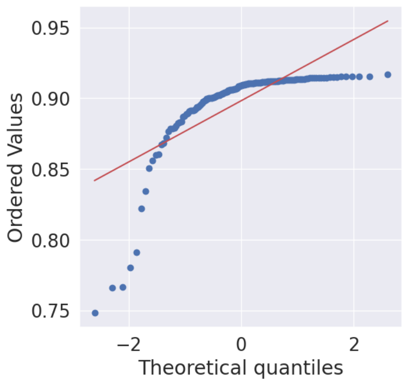
To formally assess the correlation between accuracy and power usage, we calculated Spearman's rank correlation coefficient, which showed $\rho = 0.013$ and a p-value of 0.875, more than α. This indicates that the correlation between accuracy and power usage is statistically insignificant. This can be explained by the fact that, as described, the power consumption decreases after some 30 generations.

Correlation between the number of layers and power usage

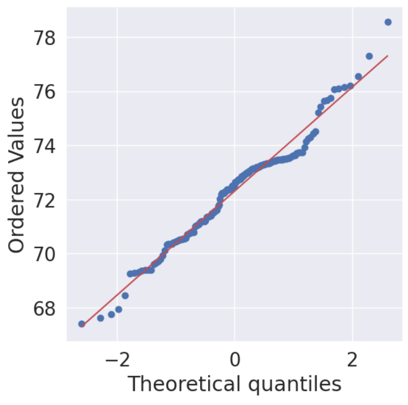
Figure 4.9 presents the number of layers and power usage evolution of each generation's best individual over 150 generations. At the beginning and until the 30th generation, the number of layers increases while accompanying an increase in power consumption. After this, the number of layers stabilizes, dropping after the 130th generation, with the power consumption decreasing slightly as well.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on the number of layers resulted in a value of 0.752 with a corresponding p-value of 1.26×10^{-14}. We reject the null hypothesis since the p-value is less than α.

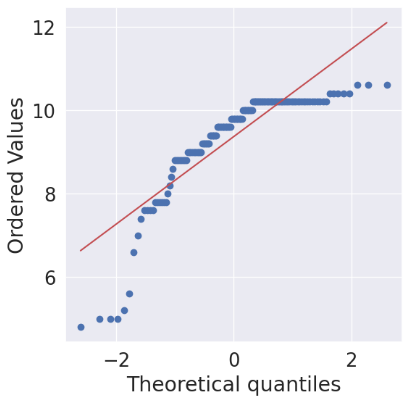
Following the assessment of normality using both Q-Q plots (Figures 4.8b and 4.8c) and the Shapiro-Wilk tests, it was determined that the data does not follow



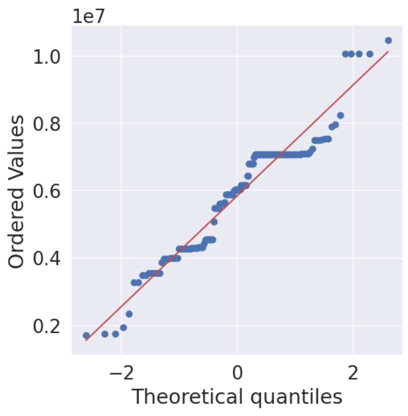
(a) Accuracy



(b) Power usage



(c) Num. of Layers



(d) Trainable Parameters

Figure 4.8: Q-Q plots showcasing the distribution comparisons for accuracy, power usage, number of layers, and number of trainable parameters over five runs of the power experiment.

a normal distribution.

To formally assess the correlation between the number of layers and power usage, we calculated the Spearman's rank correlation coefficient, which showed $\rho = 0.173$ and a p-value of 3.43×10^{-2} which is less than α. This indicates that the correlation between the number of layers and power usage is statistically significant, thus suggesting that there is a trend in how power usage increases with the number of layers.

Correlation between the number of trainable parameters and power usage

Figure 4.10 shows how the number of trainable parameters and power usage evolve over 150 generations. We can observe that these two metrics apparently follow each other. That is, when there is an increase in the number of trainable parameters, there is also an increase in power usage. This behavior is opposite to the baseline experiment, perhaps because when minimizing power consump-

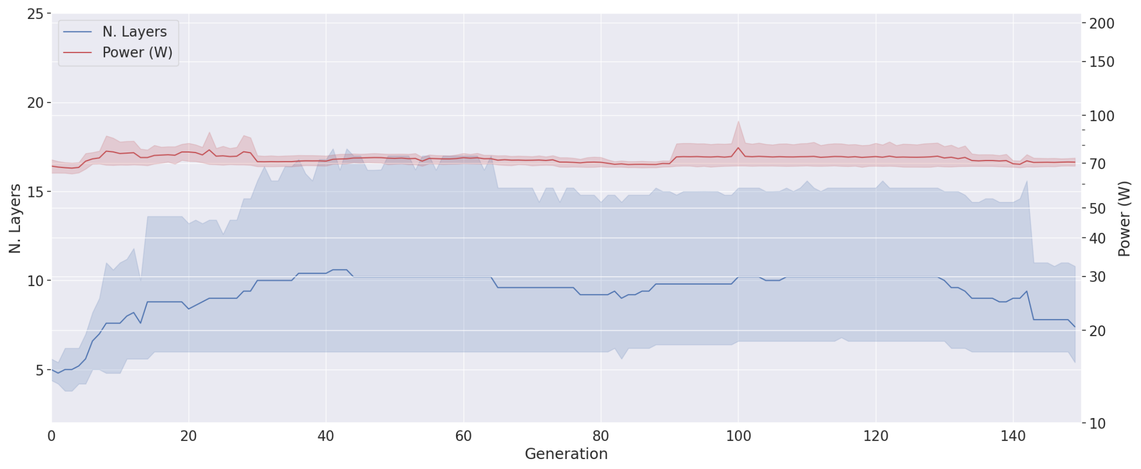


Figure 4.9: Evolution of the number of layers and power usage over 150 generations of the power experiment.

tion, the number of trainable parameters is somewhat enforced to be reduced, thus being closely linked to the power consumption.

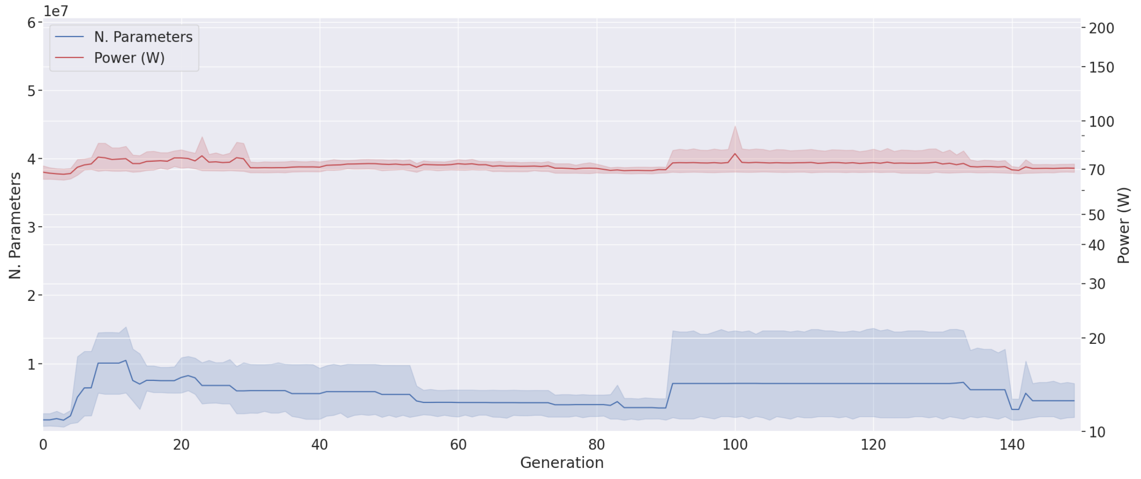


Figure 4.10: Evolution of the number of trainable parameters and power usage over 150 generations of the power experiment.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on the number of trainable parameters resulted in a value of 0.939 with a corresponding p-value of 4.57×10^{-6}. We reject the null hypothesis since the p-value is less than α.

Following the assessment of normality using both Q-Q plots (Figures 4.8b and 4.8d) and the Shapiro-Wilk tests, it was determined that the data does not follow a normal distribution.

To formally assess the correlation between the number of trainable parameters and power usage, we calculated Spearman's rank correlation coefficient, which showed $\rho = 0.844$ and a p-value of 7.29×10^{-42} which is less than α. This indicates that the correlation between the number of trainable parameters and power usage is statistically significant, thus suggesting that there is a trend in how power usage increases with the number of trainable parameters.

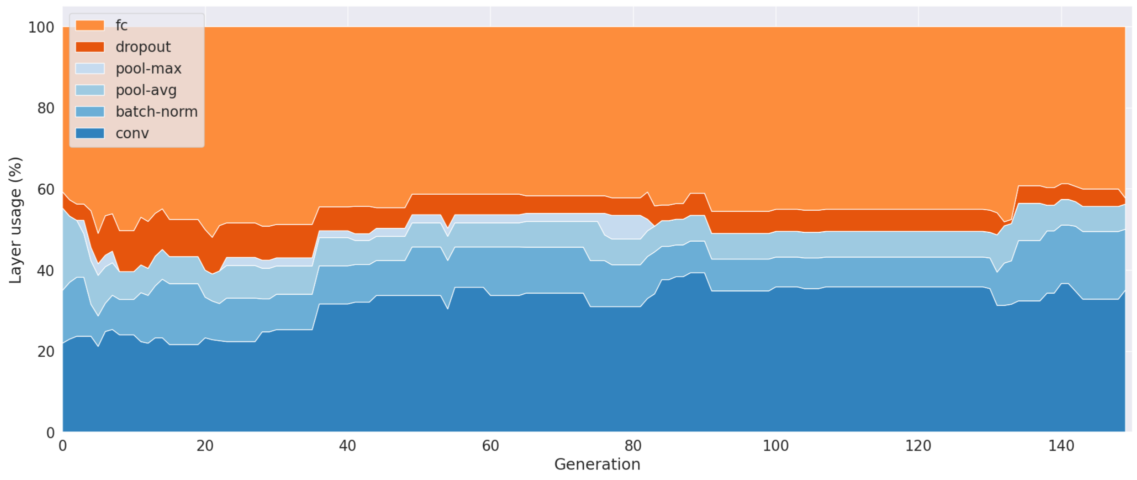
Layer composition breakdown

The evolution of layer composition of each generation's best model by percentage is presented in Figure 4.11a with Figure 4.11b presenting the power usage over 150 generations. Table 4.3 showcases the values of every 30th generation to facilitate the analysis. The first generation has a predominance of *fc* layers, followed by a similar composition of *pool-avg* and *conv* layers. After an initial increase in power usage up to the 30th generation, the most noticeable changes in composition are the increase of *fc* layers from 40.7% ± 6.0% to 49.2% ± 27.2% and a decrease of *pool-avg* layers from 20.3% ± 14.5% to 7.5% ± 10.4%. After a stable phase, there is a substantial increase in power usage, likely motivated by a slight increase in the percentage of *fc* layers. The last generation culminates in models predominated by *fc* layers, followed by *conv* layers. Noticeably, there are no *pool-max* layers and the percentage of *dropout* layers is significantly reduced, which is different from what is observed in the baseline experiment, where *dropout* layers compose around 20% of the total of layers. This behavior might be explained by the fact that partitioning the model into two models creates a regularization mechanism that makes *dropout* layers unnecessary.

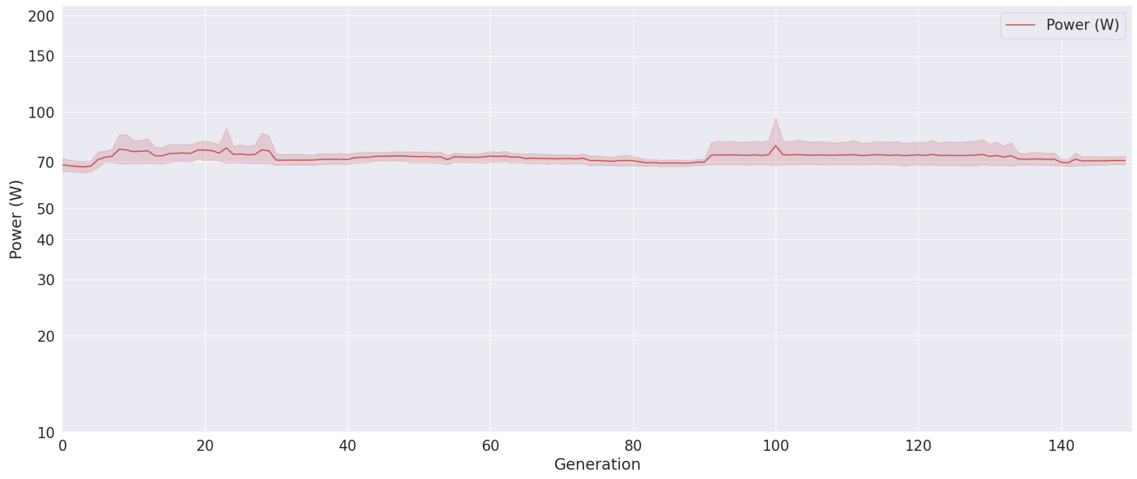
When comparing the first and last generations, we observe that both have a similar power consumption and a slight increase in the percentage of *fc* layers and a slight decrease in *batch-norm* layers. At the same time, we observe that *dropout* and *pool-avg* layers drop substantially and that the *conv* layers increase by more than 50%.

Table 4.3: Evolution of layer composition by percentage over 150 generations of the power experiment, showcasing data for every 30th generation.

Layer	**Generation**					
	1	**30**	**60**	**90**	**120**	**150**
fc	40.7% ± 6.0%	49.2% ± 27.2%	41.3% ± 28.0%	41.1% ± 26.5%	45.0% ± 29.7%	42.3% ± 23.9%
dropout	4.0% ± 8.9%	8.4% ± 9.4%	5.1% ± 7.5%	5.4% ± 9.6%	5.4% ± 9.6%	1.4% ± 3.2%
pool-max	0.0% ± 0.0%	2.0% ± 4.5%	2.0% ± 4.5%	0.0% ± 0.0%	0.0% ± 0.0%	0.0% ± 0.0%
pool-avg	20.3% ± 14.5%	7.5% ± 10.4%	5.9% ± 8.2%	6.3% ± 8.7%	6.3% ± 8.7%	6.2% ± 8.5%
batch-norm	13.0% ± 12.0%	8.1% ± 13.1%	9.9% ± 9.4%	7.8% ± 7.5%	7.3% ± 7.2%	15.0% ± 8.6%
conv	22.0% ± 22.8%	24.8% ± 19.3%	35.8% ± 23.2%	39.3% ± 28.0%	35.9% ± 32.5%	35.1% ± 23.7%



(a) Layer composition



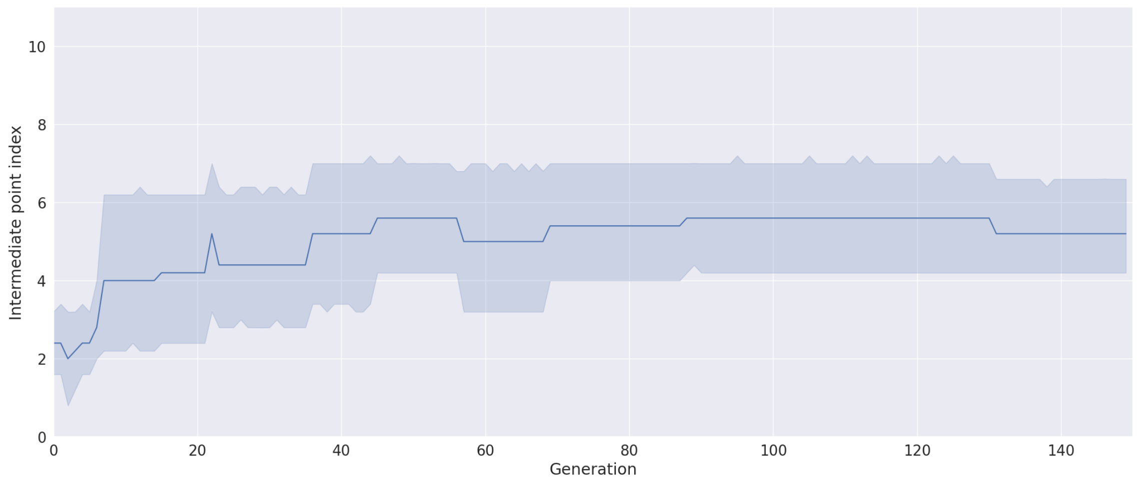
(b) Power usage

Figure 4.11: Evolution of layer composition by percentage and power consumption over 150 generations of the power experiment.

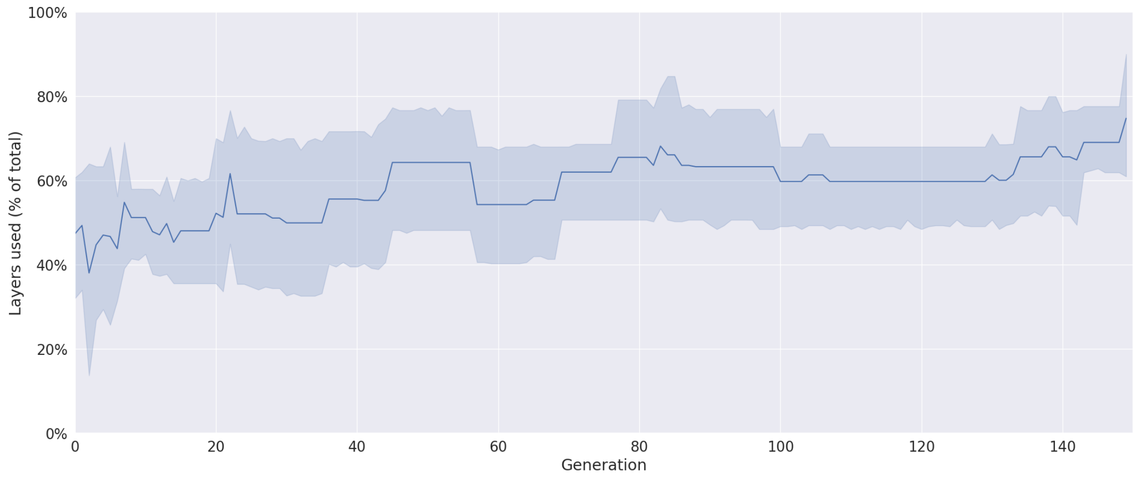
Model partitioning results

As described in Section 3.2, a new attribute was given to each individual - an intermediate point index. That point corresponds to the index of the layer where an additional output is added. After training, two models are created: one with all the layers - denominated *left* model -, and another one with the layers up to the mentioned point - denominated *right* model.

The evolution of the intermediate point index is presented in Figure 4.12a. We can observe that the index increases until generation 50, nearly stabilizing afterwards. An increase in this parameter results in a *right* model with more layers of the *left* model (as observed in Figure 4.12b). Thus, we conclude that even though the *right* model is smaller than the *left* model, it does not evolve towards a smaller model. This is likely due to the weight attributed to the power usage in the fitness function. Smaller models are more likely to be discarded due to their inferior accuracy.



(a) Intermediate Point Index



(b) Layers used by the "right" model, as a percentage of total

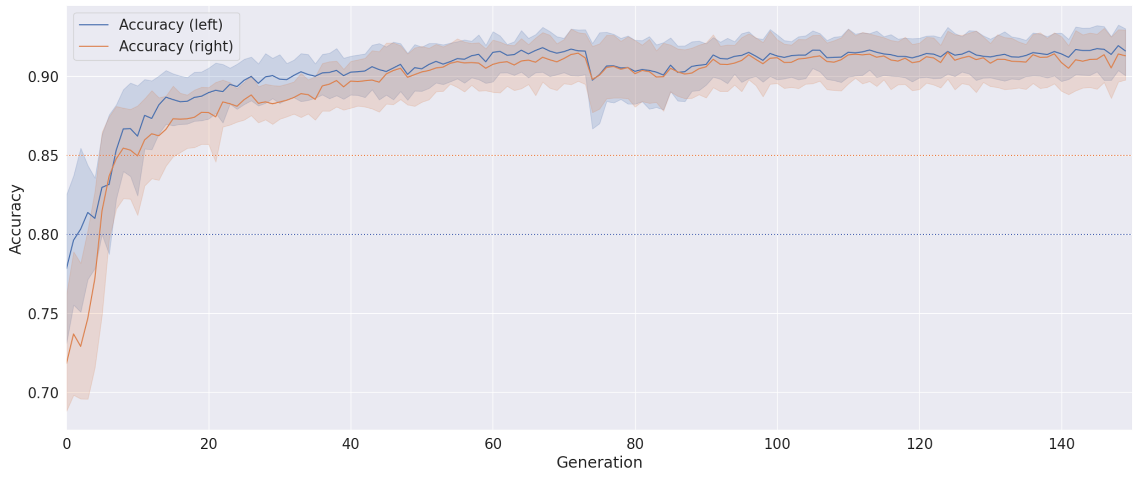
Figure 4.12: Evolution of the intermediate point index over 150 generations of the power experiment.

Figures 4.13a and 4.13b present the evolution of accuracy and power usage over 150 generations on both *left* and *right* models. The first one also includes two horizontal, dotted lines showcasing the threshold required in the fitness function, with the blue line relative to the *left* model and the orange line relative to the *right* model. When both the lines fall under or on the threshold lines, the fitness function does not consider the power usage.

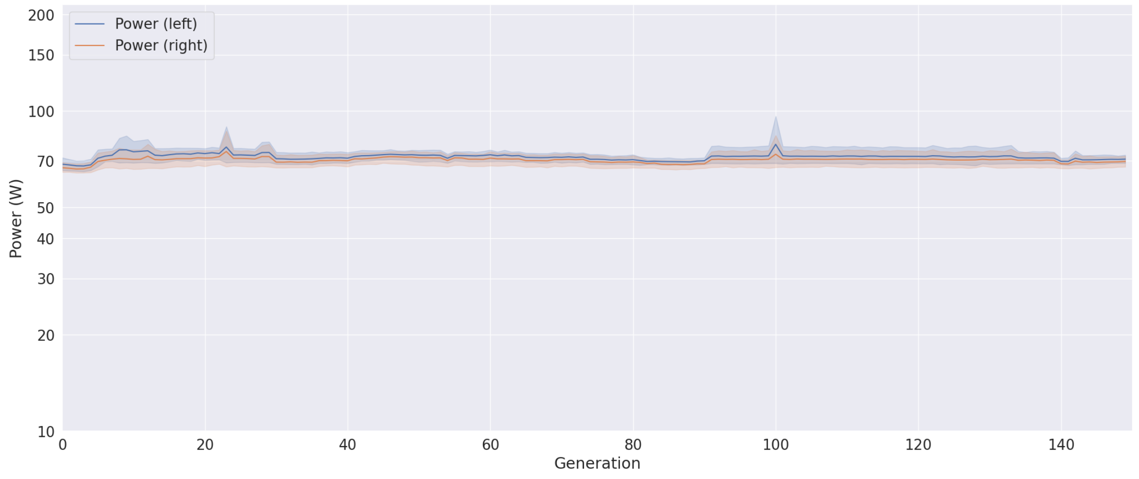
We can observe that, on average, the *right* model - the smaller model - has an inferior accuracy compared to its counterpart, even though the difference is marginal in most generations. Specifically $acc_{left} - acc_{right}$ ranges from -5.20×10^{-3} to

7.42×10^{-2} with a mean value of 7.16×10^{-3}. The negative values are relative to the cases where the accuracy of the *right* model surpasses the accuracy of the *left* model. Furthermore, this difference is larger at the beginning of the evolutionary process. It decreases afterwards due to the fact that, as seen in Figure 4.12b, the *right* model usage of layers increases, thus making it more closely related to its counterpart partition.

More importantly, the power usage of the *right* model is, on average, always inferior to the power usage of its counterpart. Specifically, $power_{left} - power_{right}$ ranges from 0.85 W to 4.70 W with a mean value of 1.53 W.



(a) Accuracy



(b) Power usage

Figure 4.13: Evolution of accuracy and power usage over 150 generations of the power experiment on both model partitions.

Summary

We conclude that there is no statistically significant correlation between accuracy and power consumption. This is because the power consumption stabilizes during most of the process, thus not exhibiting any substantial increase or decrease. We also conclude that there is a positive correlation between the number of layers and power usage and between the number of trainable parameters and power usage.

We verify that, at the end of the evolutionary process, the most predominant types of layers are *fc* and *conv* and that, notably, the percentage of *dropout* layers is low, which might be explained by the fact that partitioning the model into two models creates a regularization mechanism that makes these layers unnecessary.

We also show that partitioning the models as described results in smaller and more efficient models with comparable accuracy relative to the whole model.

4.5 Skip Connections Experiment

Similar to the Power Experiment (described in Section 4.4), this experiment utilizes the approaches described in Section 3 (except for the one that uses pre-trained models as seed, as described in Section 3.5) but allows the utilization of skip connections by having non-zero values for the *Add connection* and *Remove connection* fields. More specifically and as presented in Table 4.1, both fields have a probability of 15%. The employed fitness function is presented in Equation 3.3.

Correlation between accuracy and power usage

Figure 4.14 presents the accuracy and power usage evolution of each generation's best individual over 150 generations. Accuracy increases mainly at the beginning of the evolutionary process, accompanied by a decrease in power usage. After the 40th generation, both stabilize with accuracy slightly increasing until the last generation.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on accuracy yielded a value of 0.654 with a corresponding p-value of 2.74×10^{-17}, and on power, it resulted in a value of 0.774 with a p-value of 6.40×10^{-14}. Since, in both cases, the p-value is less than α, we reject the null hypothesis.

Following the assessment of normality using both Q-Q plots (Figures 4.15a and

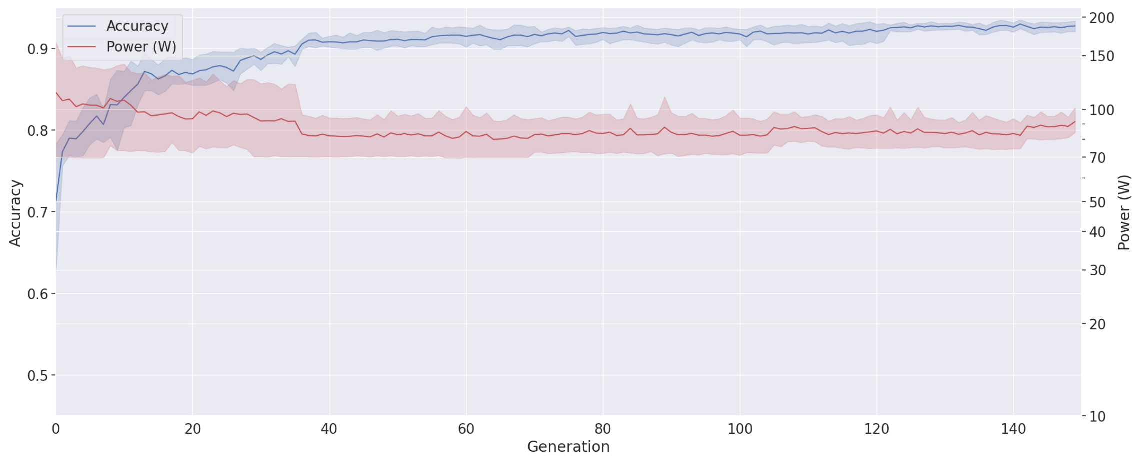


Figure 4.14: Evolution of accuracy and power usage over 150 generations of the skip connections experiment.

4.15b) and the Shapiro-Wilk tests, it was determined that the data does not follow a normal distribution.

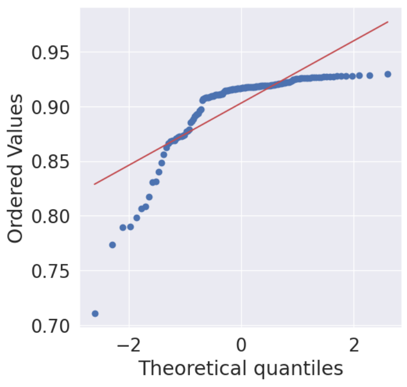
To formally assess the correlation between accuracy and power usage, we calculated Spearman's rank correlation coefficient, which showed $\rho = -0.312$ and a p-value of 1.03×10^{-4} less than α. These results collectively suggest a statistically significant negative correlation between accuracy and power usage, i.e., there is evidence to indicate that as accuracy increases, there tends to be a decrease in power usage.

Correlation between the number of layers and power usage

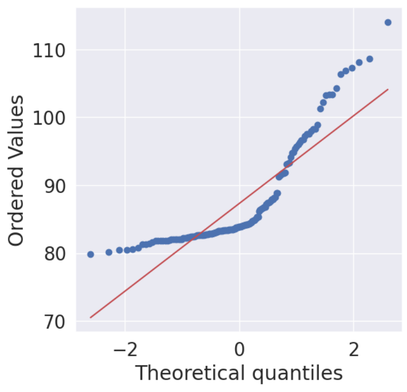
Figure 4.16 presents the number of layers and power usage evolution of each generation's best individual over 150 generations. Unlike previous experiments, an increase in the number of layers apparently does not increase the power usage. On the contrary, we can observe that the power usage drops although the number of layers increases. This is likely due to the fact that this experiment allows skip connections.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on the number of layers resulted in a value of 0.912 with a corresponding p-value of 7.00×10^{-8}. We reject the null hypothesis since the p-value is less than α.

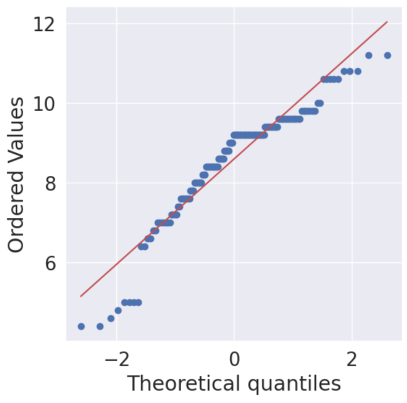
Following the assessment of normality using both Q-Q plots (Figures 4.15c and 4.15b) and the Shapiro-Wilk tests, it was determined that the data does not follow a normal distribution.



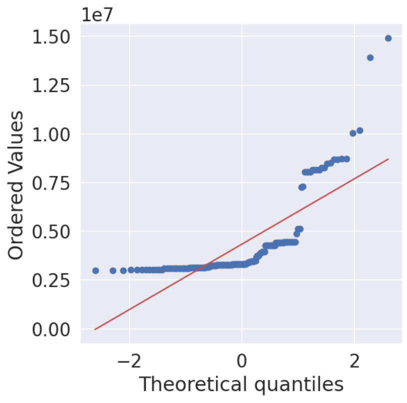
(a) Accuracy



(b) Power usage



(c) Num. of Layers



(d) Trainable Parameters

Figure 4.15: Q-Q plots showcasing the distribution comparisons for accuracy, power usage, number of layers, and number of trainable parameters over five runs of the skip connections experiment.

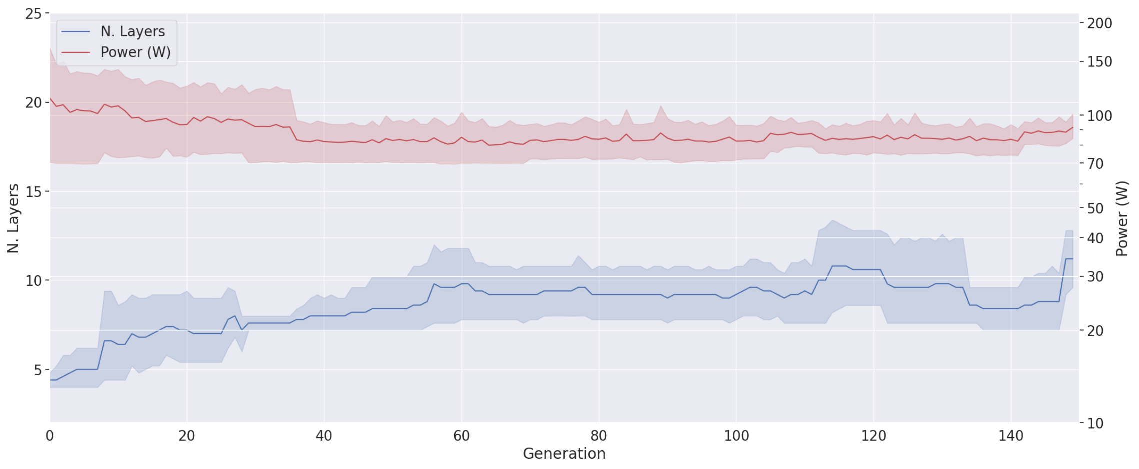


Figure 4.16: Evolution of the number of layers and power usage over 150 generations of the skip connections experiment.

To formally assess the correlation between the number of layers and power usage, we calculated the Spearman's rank correlation coefficient, which showed $\rho = -0.44$ and a p-value of 1.27×10^{-8} which is less than α. These results collectively suggest a statistically significant negative correlation between the number of layers and power usage, i.e., evidence indicates that as the number of layers increases, there tends to be a decrease in power usage.

Correlation between the number of trainable parameters and power usage

Figure 4.17 shows how the number of trainable parameters and power usage evolve over 150 generations. From the 10th generation to the 30th generation, we can observe an apparent relation between the number of trainable parameters and power usage, i.e., when the first one is relatively high, so is the other. After this, the number of trainable parameters drops and stabilizes with the power usage, presenting the same behavior.

To assess the normality of the data, we conducted a Shapiro-Wilk test. The test statistic on the number of trainable parameters resulted in a value of 0.625 with a corresponding p-value of 5.57×10^{-18}. We reject the null hypothesis since the p-value is less than α.

Following the assessment of normality using both Q-Q plots (Figures 4.15d and 4.15b) and the Shapiro-Wilk tests, it was determined that the data does not follow a normal distribution.

To formally assess the correlation between the number of trainable parameters and power usage, we calculated Spearman's rank correlation coefficient, which

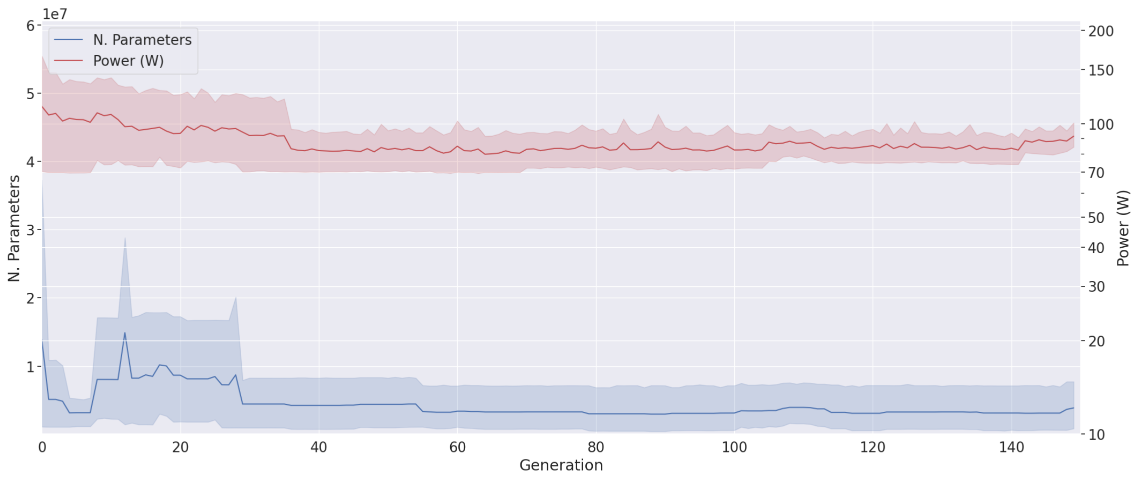


Figure 4.17: Evolution of the number of trainable parameters and power usage over 150 generations of the skip connections experiment.

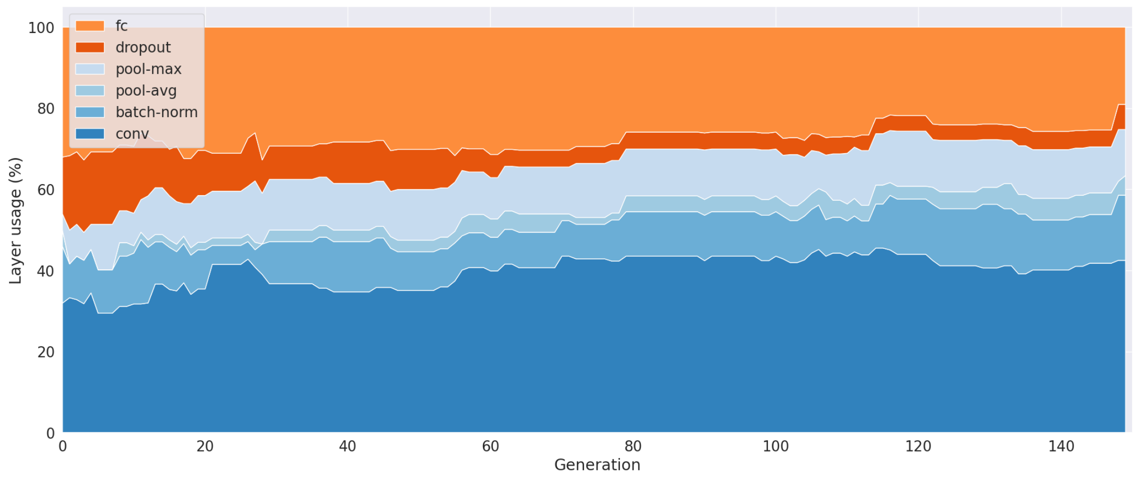
showed $\rho = 0.37$ and a p-value of 4.16×10^{-6}, less than α. This indicates that the correlation between the number of trainable parameters and power usage is statistically significant, thus suggesting that there is a trend in how power usage increases with the number of trainable parameters.

Layer composition breakdown

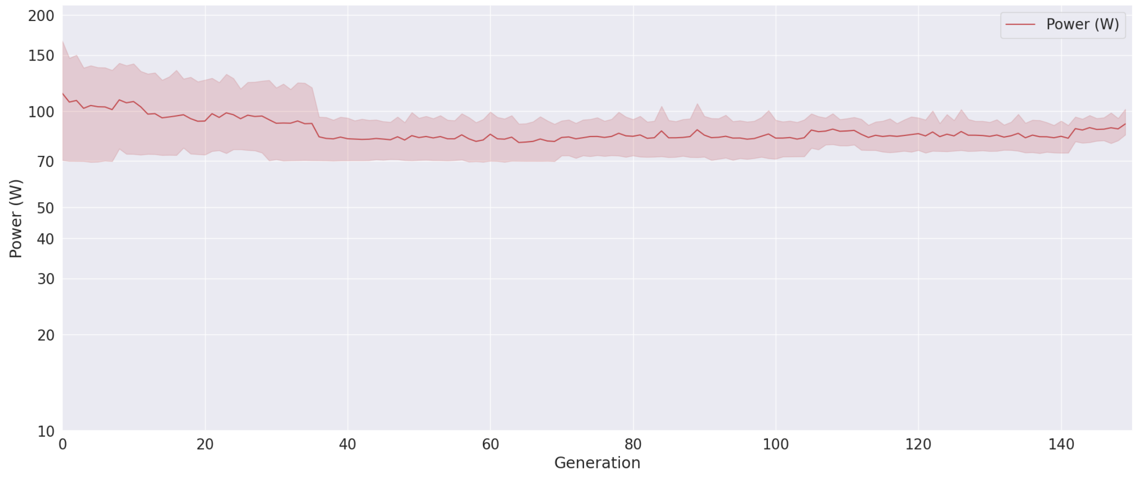
The evolution of layer composition of each generation's best model by percentage is presented in Figure 4.18a with Figure 4.18b presenting the power usage over 150 generations. Table 4.4 showcases the values of every 30th generation to facilitate the analysis.

At the beginning of the evolutionary process, the power consumption shows an accentuated decrease, after which it stabilizes. To analyze the sudden decrease after the 30th generation, we can compare the layer composition of that generation to the layer composition of the 60th generation. There is a decrease of *dropout*, *pool-max*, and *batch-norm* layers and an increase of *conv* layers. As the process continues, there is a noticeable decrease in the percentage of *fc* layers and a noticeable increase of *batch-norm* layers. The last generation culminates in models where *conv* layers are predominant, accounting for 42.5% of the layers. This is followed by *fc* and *batch-norm* layers, which contribute 19% and 16%, respectively.

When comparing the first and last generations, we observe that the last one has a relatively reduced power consumption and a significant increase in the percentage of *conv* and *pool-max* layers. There was also a significant decrease in *fc* and *dropout* layers. This decrease in *dropout* layers might have a similar explanation as the one of the Power Experiment: partitioning the model into two models creates a regularization mechanism that renders these layers unnecessary.



(a) Layer composition



(b) Power usage

Figure 4.18: Evolution of layer composition by percentage and power consumption over 150 generations of the skip connections experiment.

Model partitioning results

As described in Section 3.2, a new attribute was given to each individual - an intermediate point index. That point corresponds to the index of the layer where an additional output is added. After training, two models are created: one with all the layers - denominated *left* model -, and another one with the layers up to the mentioned point - denominated *right* model.

The evolution of the intermediate point index is presented in Figure 4.19a. We can observe that the index increases until generation 60, not having substantial

Table 4.4: Evolution of layer composition by percentage over 150 generations of the skip connections experiment, showcasing data for every 30th generation.

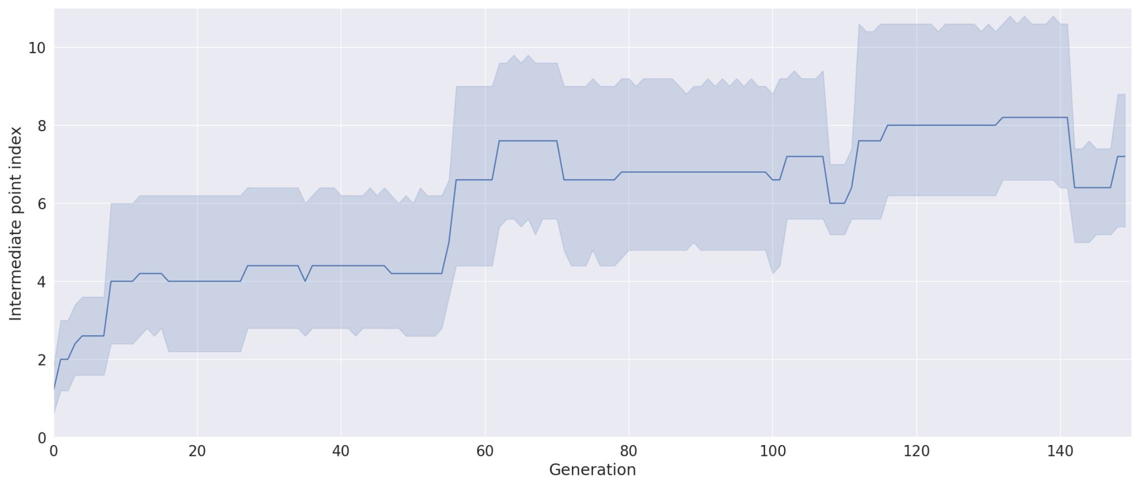
Layer	**Generation**					
	1	**30**	**60**	**90**	**120**	**150**
fc	32.0% ± 12.5%	29.3% ± 7.7%	30.0% ± 12.9%	25.8% ± 13.9%	21.8% ± 15.0%	19.0% ± 12.2%
dropout	14.0% ± 12.9%	8.2% ± 12.6%	5.7% ± 5.5%	4.2% ± 5.9%	3.8% ± 5.6%	6.2% ± 6.2%
pool-max	4.0% ± 8.9%	12.5% ± 12.5%	10.4% ± 10.3%	11.6% ± 12.6%	13.6% ± 18.7%	11.3% ± 17.6%
pool-avg	4.0% ± 8.9%	2.9% ± 6.4%	4.5% ± 6.5%	3.9% ± 5.4%	3.2% ± 4.4%	4.9% ± 6.8%
batch-norm	14.0% ± 12.9%	10.4% ± 10.6%	8.6% ± 5.5%	11.0% ± 8.2%	13.6% ± 11.9%	16.0% ± 9.9%
conv	32.0% ± 12.5%	36.8% ± 10.2%	40.8% ± 8.3%	43.6% ± 6.9%	44.0% ± 10.6%	42.5% ± 14.8%

increases or decreases afterward. An increase in this parameter results in a *right* model with more layers of the *left* model (as observed in Figure 4.19b). Thus, we conclude that even though, on average, the *right* model is smaller than the *left* model, it does not evolve towards an even smaller model. This is likely due to the weight attributed to the power usage in the fitness function. Smaller models are more likely to be discarded due to their inferior accuracy.

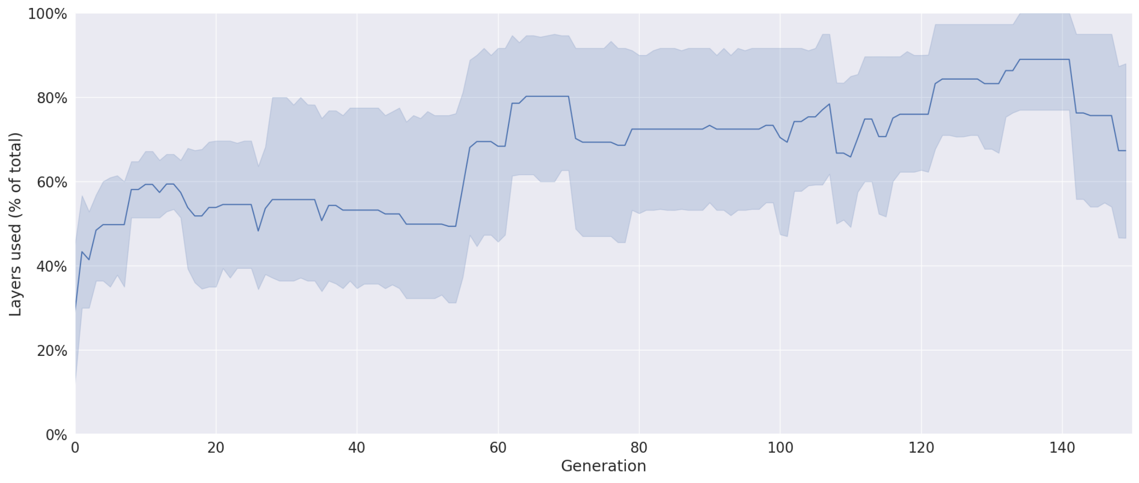
Figures 4.20a and 4.20b present the evolution of accuracy and power usage over 150 generations on both *left* and *right* models. The first one also includes two horizontal, dotted lines showcasing the threshold required in the fitness function, with the blue line relative to the *left* model and the orange line relative to the *right* model. When both the lines fall under or on the threshold lines, the fitness function does not consider the power usage.

Generally, we can observe that, on average, the *right* model - the smaller model - has an inferior accuracy compared to its counterpart, even though the difference is marginal in most generations, emphasizing the generations posterior to the 70th generation. Specifically $acc_{left} - acc_{right}$ ranges from -1.79×10^{-2} to 2.31×10^{-2} with a mean value of 4.40×10^{-3}. The negative values are relative to the cases where the accuracy of the *right* model surpasses the accuracy of the *left* model. The difference between the two models in terms of accuracy decreases during the evolutionary process since the *right* model usage of the total number of layers increases as seen in Figure 4.19b.

More importantly, the power usage of the *right* model is, on average, always inferior to the power usage of its counterpart. Specifically, $power_{left} - power_{right}$ ranges from -8.63 W to 15.66 W with a mean value of 0.79 W. The negative values mean that the *right* model obtained a larger power usage than the *left* model.



(a) Intermediate Point Index



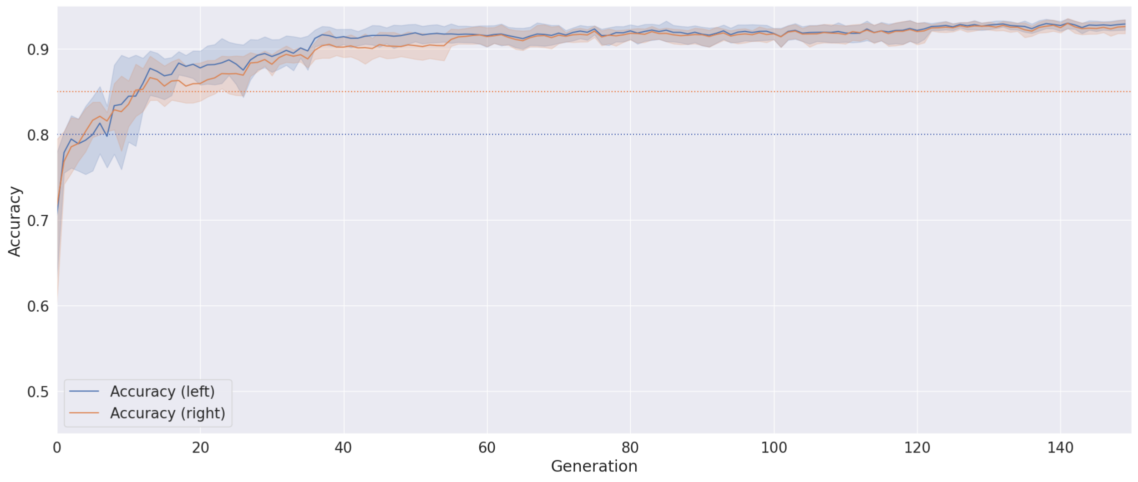
(b) Layers used by the "right" model, as a percentage of total

Figure 4.19: Evolution of the intermediate point index over 150 generations of the skip connections experiment.

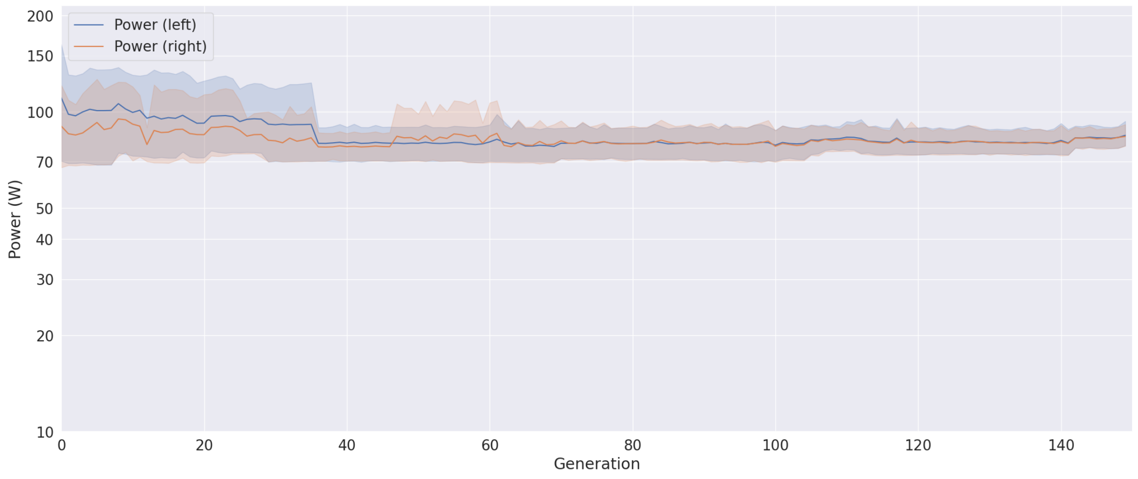
Summary

We conclude that there is a negative correlation between accuracy and power usage and between the number of layers and power usage. The existence of a correlation between accuracy and power usage is contrary to the Power Experiment where no correlation is shown. This happens since, in this experiment, the initial generations present higher power consumption and end up showing a significant reduction, thus presenting a decreasing trend. We also conclude there is a positive correlation between the number of trainable parameters and power usage.

We verify that, at the end of the evolutionary process, the most predominant types of layers are *conv* at 42.5%, followed by *fc* and *batch-norm* at 19% and 16%,



(a) Accuracy



(b) Power usage

Figure 4.20: Evolution of accuracy and power usage over 150 generations of the skip connections experiment on both model partitions.

respectively.

We also show that partitioning the models as described results in a smaller and more efficient model with comparable accuracy relative to the complete model.

4.6 Experiments Comparison

This section will compare the three experiments through statistical tests, line charts, and other relevant statistical values.

We consider five groups: one for the **baseline** experiment (Section 4.3), two for the **power** experiment (Section 4.4), and another two for the **skip** (connections) experiment (Section 4.5). In the latter two experiments, we consider the measurements of the accuracy of the partitioned models: *left* and *right*. Each group corresponds to the mean value of a measured metric over the five experimental runs for each generation.

4.6.1 Accuracy

To assess the normality of the data, we conducted a Shapiro-Wilk test where the null hypothesis (H_0) is that the data follows a normal distribution and the alternative hypothesis (H_1) is that the data does not follow a normal distribution. The results are presented in Table 4.5. From these, we can conclude that the data does not follow a normal distribution since, for every case, the p-value is less than α.

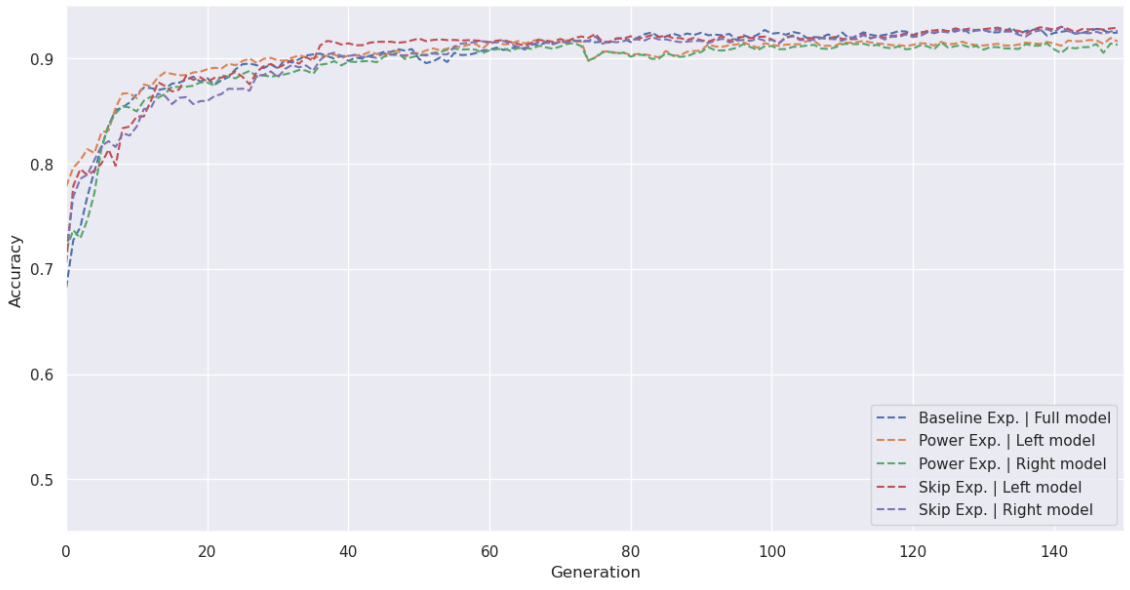
Table 4.5: Shapiro-Wilk test on accuracy of the three experiments.

Experiment	Metric	Test Statistic	P-value	Reject H_0
Baseline	Accuracy	0.601	1.64×10^{-18}	Yes
Power	$Accuracy_{left}$	0.593	1.09×10^{-18}	Yes
	$Accuracy_{right}$	0.545	1.12×10^{-19}	Yes
Skip	$Accuracy_{left}$	0.612	2.91×10^{-18}	Yes
	$Accuracy_{right}$	0.692	2.45×10^{-16}	Yes

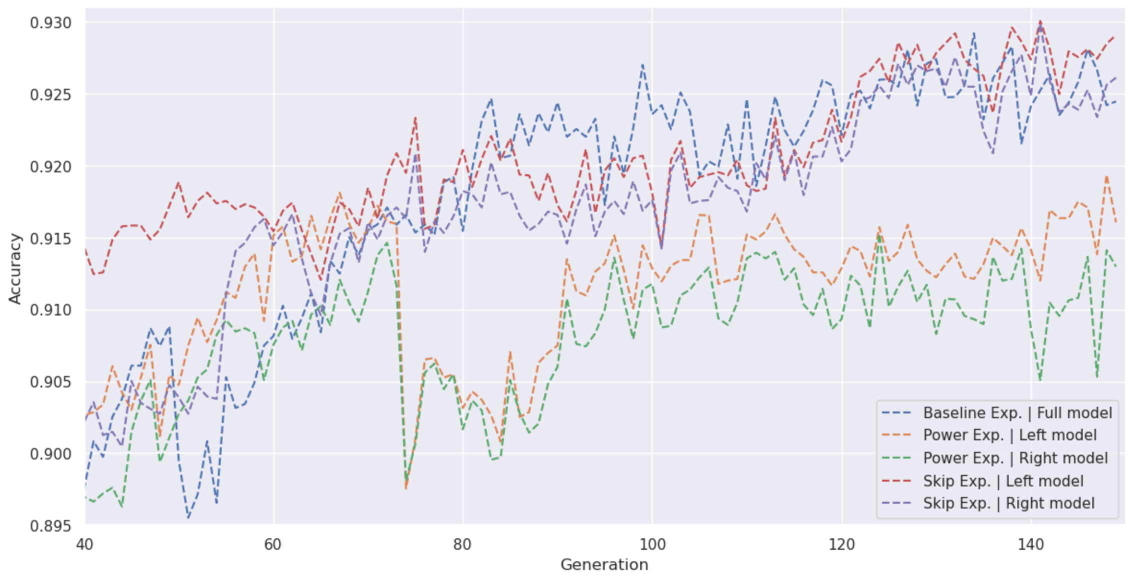
Since there are more than two groups, neither follow a normal distribution, and since the samples are independent, we performed the Kruskal-Wallis test. We obtained a statistic value of 99.49 and a p-value of 1.26×10^{-20}, which is inferior to α. Therefore, we reject H_0 and conclude that the median of at least one group is different from the others.

Having confirmed substantial differences between the groups, we performed the Mann-Whitney U test post-hoc between groups and then adjusted the p-values using Bonferroni correction. The results from this test are presented in Table 4.6 with the bold values denoting statistically significant differences, i.e., the cases where the p-value is less than α, which suggest that the differences observed in those cases are unlikely to have occurred by random change and should be better analyzed.

Figures 4.21a and 4.21b show the comparison between the five mentioned groups. That is the evolution of accuracy over 150 generations on the three experiments, with the accuracy of partitioned models shown as well, when applicable. The latter, Figure 4.21b, focuses on the evolution after the 40th generation, showcasing the range $[40, 150]$ on the X axis and the range $[0.895, 0.931]$ on the Y axis.



(a) Full view



(b) Close up view from generation 40 to 150

Figure 4.21: Evolution of the accuracy over 150 generations on the three experiments.

Table 4.6: Pair-wise comparison of used groups on accuracy metric, using Mann-Whitney U post-hoc test with Bonferroni correction with bold values denoting statistically significative differences.

			Experiment				
			Baseline	**Power**		**Skip**	
		Metric	Accuracy	$Accuracy_{left}$	$Accuracy_{right}$	$Accuracy_{left}$	$Accuracy_{right}$
Experiment	**Baseline**	Accuracy					
	Power	$Accuracy_{left}$	$\mathbf{3.62 \times 10^{-4}}$				
		$Accuracy_{right}$	$\mathbf{3.82 \times 10^{-7}}$	$\mathbf{3.87 \times 10^{-4}}$			
	Skip	$Accuracy_{left}$	1.00	$\mathbf{2.49 \times 10^{-12}}$	$\mathbf{4.99 \times 10^{-17}}$		
		$Accuracy_{right}$	1.00	$\mathbf{9.63 \times 10^{-5}}$	$\mathbf{2.48 \times 10^{-8}}$	0.111	

Table 4.7 presents, for each experiment and for each of the two partitioned models (when applicable), the mean value, the standard deviation, the median, and the difference between the median value and the median value of the baseline experiment.

Table 4.7: Mean value, standard deviation, median and difference to baseline median of the experiments accuracy.

Experiment	**Metric**	**Mean**	**SD**	**Median**	**Diff. to Baseline**
Baseline	Accuracy	0.904	0.037	0.916	
Power	$Accuracy_{left}$	0.902	0.024	0.911	-0.005
	$Accuracy_{right}$	0.895	0.034	0.907	-0.009
Skip	$Accuracy_{left}$	0.905	0.035	0.918	0.002
	$Accuracy_{right}$	0.901	0.035	0.916	0.000

Analyzing Table 4.6, we can conclude that there are statistically significative differences between the accuracy of the baseline experiment and all measured accuracies of the power experiment; all measured accuracies of the power experiment and all measured accuracies of the skip experiment; and, in the power experiment, there is a statistically significative difference between the accuracy of the *left* model and the accuracy of the *right* model. Contrarily, there are no statistically significant differences between the baseline experiment and the skip experiment's measured accuracies and, in the skip experiment, between the accuracy of the *left* model and the accuracy of the *right* model.

At the beginning, the *left* model of the power experiment has the highest accuracy value up until generation 35, where it is surpassed by the *left* model of the skip experiment. After this, between the 82nd and the 120th generations, the baseline experiment obtains the highest accuracy value. It is later overtaken by the *left* model of the skip experiment, which is the most accurate on average.

Interestingly, after the 74th generation, the formation of two groups is most noticeable: one composed of the two models of the power experiment and the other composed of the remaining lines, with the former obtaining higher accuracy values than the latter. These differences are corroborated by the fact that there are statistically significant differences between the power experiment and the other two experiments.

Although marginal, there is also a noticeable difference between the measured accuracies of the power experiment's two models, with the *left* model achieving higher values. This is confirmed by the fact that there are statistically significant differences between the measured values on the two models of this experiment.

With this information, we can conclude that the baseline and skip experiments present better accuracy than the power experiment, having no statistically significant differences. We can also conclude that, in the power experiment, the *left* model is more accurate than the *right* model.

4.6.2 Power Usage

To assess the normality of the data, we conducted a Shapiro-Wilk test where the null hypothesis (H_0) is that the data follows a normal distribution and the alternative hypothesis (H_1) is that the data does not follow a normal distribution. The results are presented in Table 4.8. From these, we can conclude that the data does not follow a normal distribution since, for every case, the p-value is less than α.

Table 4.8: Shapiro-Wilk test on power usage of the three experiments.

Experiment	Metric	Test Statistic	P-value	Reject H_0
Baseline	Power	0.893	5.77×10^{-9}	Yes
Power	$Power_{left}$	0.937	3.07×10^{-6}	Yes
	$Power_{right}$	0.901	1.51×10^{-8}	Yes
Skip	$Power_{left}$	0.699	3.80×10^{-16}	Yes
	$Power_{right}$	0.814	1.59×10^{-12}	Yes

Since there are more than two groups, neither follow a normal distribution and since the samples are independent, we performed the Kruskal-Wallis test. We obtained a statistic value of 494.53 and a p-value of 1.02×10^{-105}, which is inferior to α. Therefore, we reject H_0 and conclude that the median of at least one group is different from the others.

Having confirmed substantial differences between the groups, we performed the Mann-Whitney U test post-hoc between groups and then adjusted the p-values using Bonferroni correction. The results from this test are presented in Table 4.9 with the bold values denoting statistically significant differences, i.e., the cases where the p-value is less than α, which suggest that the differences observed in those cases are unlikely to have occurred by random change and should be better analyzed.

Table 4.9: 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.

			Experiment				
			Baseline	**Power**		**Skip**	
		Metric	Power	$Power_{left}$	$Power_{right}$	$Power_{left}$	$Power_{right}$
Experiment	**Baseline**	Power					
	Power	$Power_{left}$	$\mathbf{9.06 \times 10^{-29}}$				
		$Power_{right}$	$\mathbf{2.95 \times 10^{-31}}$	$\mathbf{1.22 \times 10^{-18}}$			
	Skip	$Power_{left}$	$\mathbf{3.23 \times 10^{-15}}$	$\mathbf{1.22 \times 10^{-49}}$	$\mathbf{1.08 \times 10^{-49}}$		
		$Power_{right}$	$\mathbf{1.15 \times 10^{-18}}$	$\mathbf{1.32 \times 10^{-49}}$	$\mathbf{1.08 \times 10^{-49}}$	0.204	

Figure 4.22 shows the comparison between the five mentioned groups. That is, the evolution of power usage over 150 generations in the three experiments with the power usage of partitioned models shown as well, when applicable.

Table 4.10 presents, for each experiment and each of the two partitioned models (when applicable), the mean value, the standard deviation, the median, and the difference between the median value and the median value of the baseline experiment.

Table 4.10: Mean value, standard deviation, median, and difference to baseline median of the experiments' power usage.

Experiment	**Metric**	**Mean**	**SD**	**Median**	**Diff. to Baseline**
Baseline	Power	97.80 W	18.84 W	99.89 W	
Power	$Power_{left}$	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
Skip	$Power_{left}$	84.33 W	7.40 W	80.39 W	-19.50 W
	$Power_{right}$	83.54 W	4.92 W	81.44 W	-18.45 W

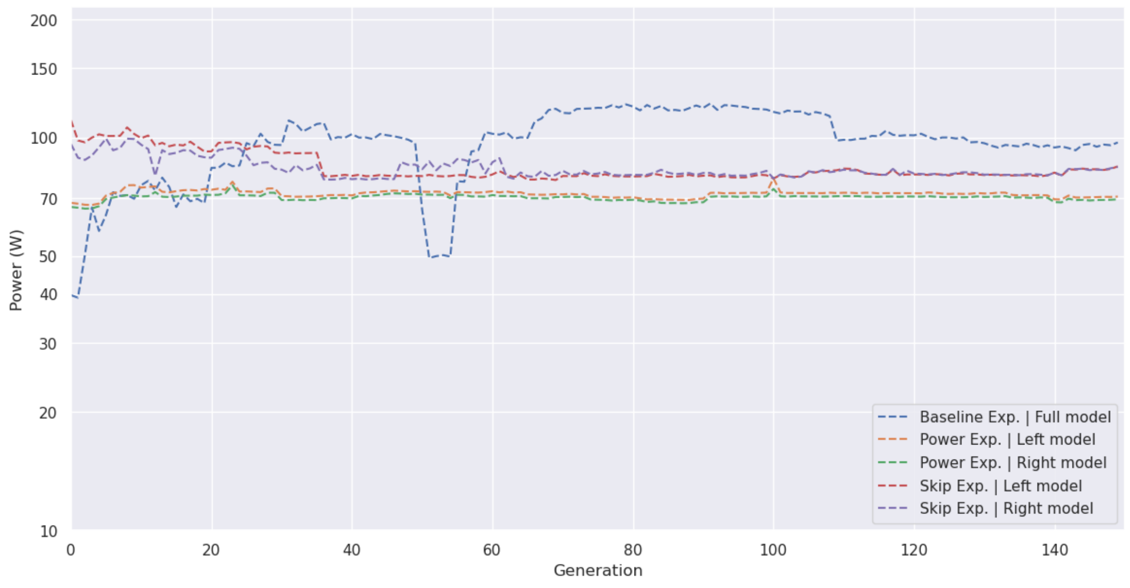


Figure 4.22: Evolution of the power usage over 150 generations on the three experiments.

Analyzing Table 4.9, we can conclude that there are statistically significative differences between all experiments except for, in the skip experiment, between the power usage of the *left* model and the power usage of the *right* model.

In the beginning, we can observe that the skip experiment presents the highest power usage, decreasing after the 35th generation and stabilizing soon after. On the contrary, we observe a spike in the power usage of the baseline experiment, rising from 40 W to 110 W in the 35th generation. Except for the steep decrease around the 50th generation, which corresponds most likely to an error, the baseline experiment has the highest power usage values until the end of the process. After the 60th generation, the power experiment always has the lowest power usage. These differences are corroborated by the fact that there are statistically significant differences between all experiments except in the case of the skip experiment's two models, which, as the figure shows, almost overlap after the 100th generation.

With this information, we can conclude that the power experiment presents the best (lowest) power usage compared to the other experiments and that its *right* model achieves, on average, the lowest power usage of all. The skip experiment presents lower power usage than the baseline experiment with statistically significant differences. Contrarily to the power experiment, there are no statistically significant differences between the measurements of its two partitioned models on the skip experiment.

4.6.3 Layer Composition

Table 4.11 showcases the comparison between the median values of each layer type on all experiments over 5 runs. As previously concluded, the baseline experiment is the most power-intensive experiment with a consumption of 99.89 W, followed by the skip experiment with a consumption of 80.39 W on its *left* partition, and by the power experiment - the most power-efficient - with a consumption of 72.20 W on its *left* partition.

Table 4.11: Median comparison of layer types in all experiments over 5 runs.

	Experiment		
Layer	Baseline	Power	Skip
conv	25.3%	33.8%	41.1%
pool-avg	3.3%	6.3%	3.8%
pool-max	2.2%	0.0%	11.6%
batch-norm	17.5%	9.4%	10.8%
dropout	15.0%	5.4%	4.3%
fc	34.9%	44.7%	27.9%

Relatively to the *conv* layer, we observe a higher prevalence in the power and in the skip experiments relative to the baseline.

The *dropout* layers have lower median percentages on the power and skip experiments, likely due to them relying on other mechanisms for regularization, such as the strategy for model partitioning, which might affect regularization. The *fc* layer type is mainly used in the power experiment with a median percentage of almost fifty percent. Contrarily, the skip experiment uses only 27.9% of these layers, likely because skip connections are performed in it, thus requiring less of these layers. Interestingly, pooling layers have a reduced percentage in most cases, except the skip experiment, where the layers consist of 3.8%*pool-avg* and 11.6% *pool-max*, showing that this experiment relies more on the mechanisms these types of layers provide. The baseline experiment uses more *batch-norm* layers than the other two experiments, showcasing that to have more efficient models, there needs to be fewer of these layers due to the number of operations they perform when normalizing data.

4.6.4 Conclusions

We conclude that the baseline and skip experiments are more accurate than the power experiment with statistical significance. Considering the median values, the *left* and *right* models of the power experiment are 0.5% and 0.9% worse than the baseline experiment, respectively. The baseline and skip experiments have no statistically significant differences. The latter shows a median accuracy 0.2% higher than the baseline on its *left* model and an identical median accuracy on its *right* model compared with the baseline.

Concerning power usage, we conclude that the power experiment presents the lowest power usage. Specifically, its *right* model has the lowest power usage of all, followed by its counterpart, the *left* model. Considering the median values, the *left* and *right* models of the power experiment consume 27.69 W and 29.18 W less than the baseline experiment, respectively. The skip experiment also presents a significantly inferior power usage than the baseline. Its *left* and *right* models consume 19.50 W and 18.45 W less, respectively.

Even though the power experiment trades off accuracy for lower power usage, the traded-off accuracy is marginal compared to the decrease in power usage. Similarly, the skip experiment achieves higher or similar accuracy scores even though it uses less power than the baseline but more power than the power experiment.

Chapter 5

Visualization

5.1 NeuroView

When conducting experiments on NE, tracking the results over the generations and the various runs of the experiment is essential. This allows researchers to comprehensively understand the evolutionary progress and assess which strategies work better.

We introduce NeuroView, a tool that enables interactive visualization of data from NE frameworks, which is currently tailored to Fast-DENSER. Through its interactive interface, this tool enhances our ability to gain insights from experiments conducted on Fast-DENSER. It supports comparing up to two metrics of a single experiment, comparing two different experiments, and examining an individual's data within an experiment, among other functionalities.

In the context of our work, this tool allowed us to readily analyze the results from experiments without requiring the programmatic creation of charts with tools such as Python's matplotlib, thus enhancing our ability to analyze and discuss results.

An example usage of NeuroView is shown in Figure 5.3, where the average value of fitness and power consumption evolution over 150 generations on five runs is compared. Any adjustments made to the controls lead to instantaneous updates in the displayed charts.

NeuroView was demoed at the first all-hands meeting of NextGenAI - Centre For Responsible AI at Sword Health's headquarters in Porto on June 29th, 2023.

5.1.1 Architecture

NeuroView is a web app with a Python backend developed with Flask and a Javascript frontend developed with React and its ecosystem. As shown in Figure 5.1, the backend server gets the experimental results from the server where they are executed, and it sends the experimental settings and results to the frontend after being requested to do so. The end-user accesses the NeuroView frontend through the HTTP port. NeuroView can be deployed in a single Docker container where its execution is managed through *gunicorn*, a Python WSGI HTTP server.

For simplification and efficiency purposes, the results can be periodically cached in the NeuroView machine with tools such as *rsync* to prevent redundant requests of the same resources.

All data manipulation is performed directly on the frontend to minimize the number of requests from the frontend to the backend and the execution time after modifying any controls.

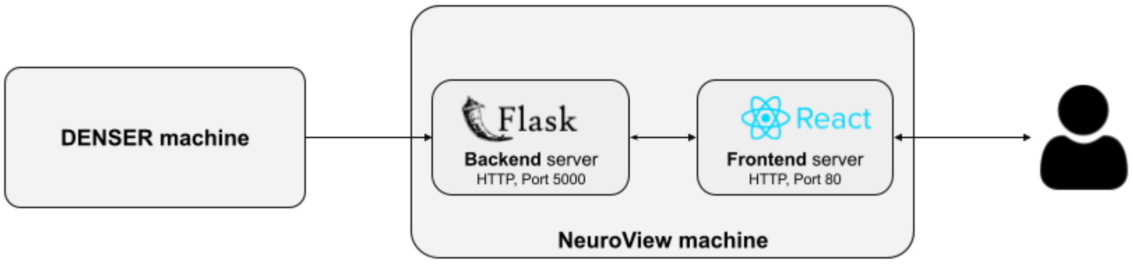


Figure 5.1: NeuroView architecture.

5.1.2 Features

Experiment visualization

The core functionality of NeuroView, the basic visualization of results (Figure 5.2), is presented on a chart that the user's controls can modify. The X axis is relative to the number of generations in all cases. The Y axis is relative to the selected metric on the chart controls when two metrics of different scales are selected (e.g., fitness and power, as shown in Figure 5.3), and an additional Y axis is shown relative to the second metric.

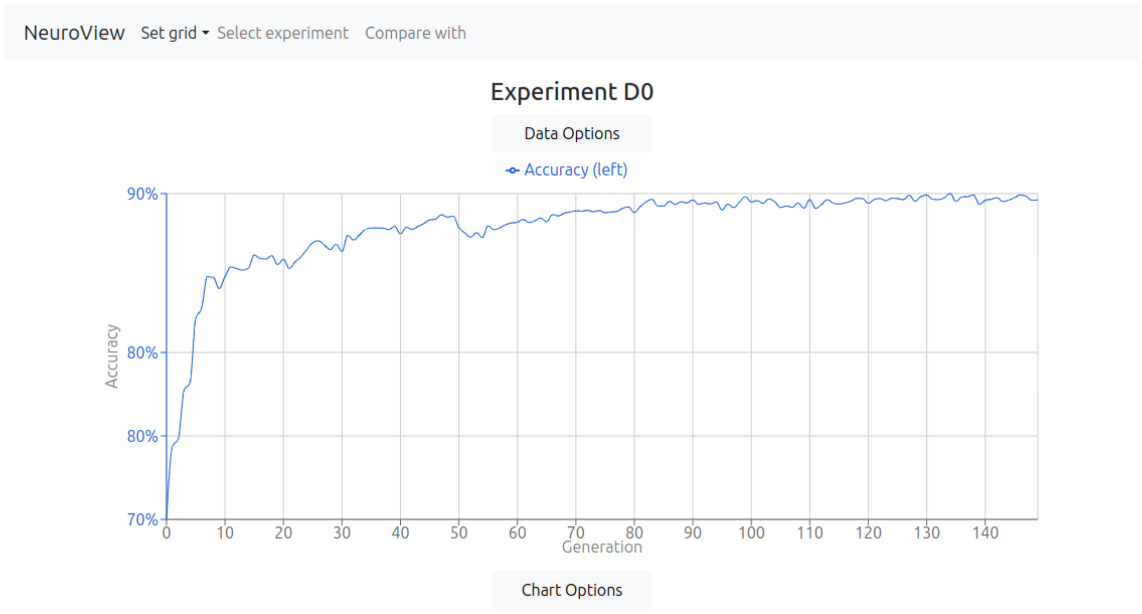


Figure 5.2: Example visualization of results on NeuroView with a single chart.

Setting grid size

This tool (Figure 5.4) allows the user to define the number of chart blocks shown on the page. When expanding the grid size, existing charts maintain their positions and settings. On the other hand, reducing the grid size deletes any charts that exist on the removed positions.

Data controls

The data controls widget (Figure 5.5a) shows the number of runs an experiment has and how many completed generations each run has and allows the user to select which generations shall be considered. It also permits the user to decide if the overall data should consider every individual of a generation or if only each generation's best individual by fitness. If one experiment is being compared with another, the widget shows controls for the data of both experiments.

Chart controls

The chart controls widget (Figure 5.5b) is associated with each shown chart. It allows the selection of up to two metrics when a single experiment is being analyzed or of one metric if one is being compared with another. The *Sync ID* field can be input with an arbitrary string that allows the synchronization of two or more charts; that is, when hovering over one of the synced charts, a pointer will

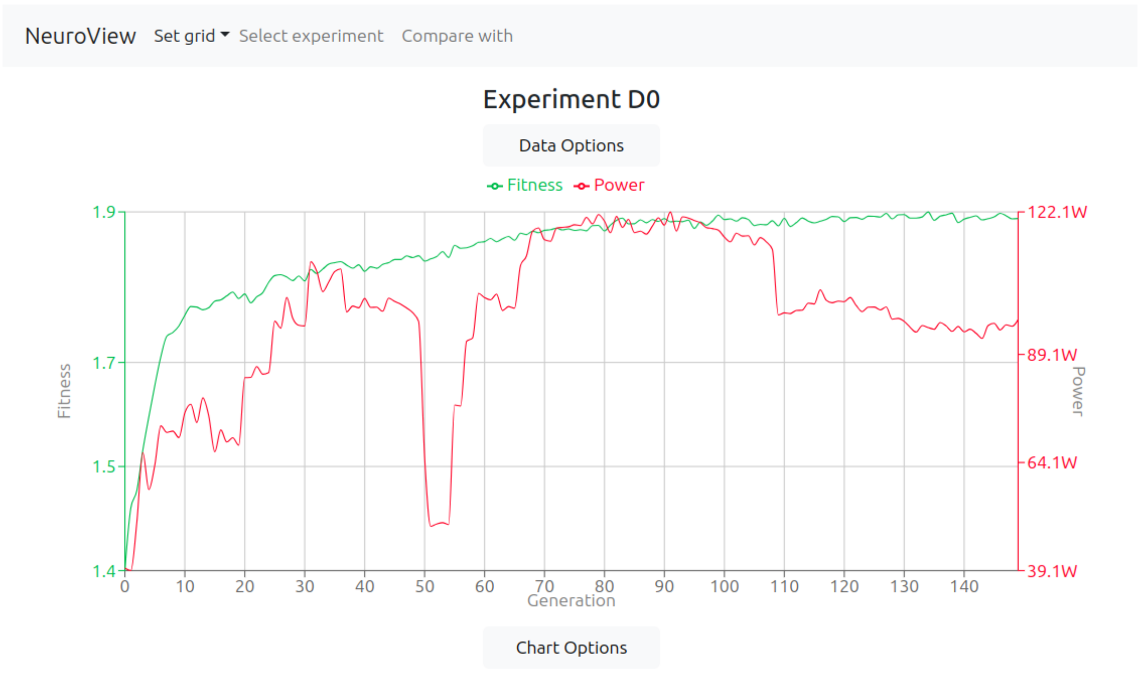


Figure 5.3: Example visualization of results on NeuroView with a single chart and two metrics.

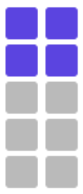


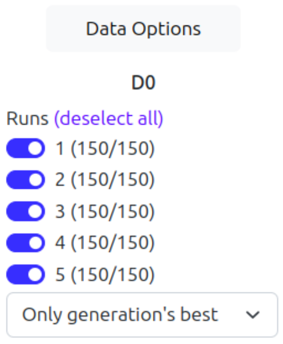
Figure 5.4: Grid size selector that allows control of the number of charts.

be shown on the other ones, allowing for a faster comparison of data. When activated, the *Tighten* select field modifies the domain on the Y-axis to range from the minimum to the maximum value instead of showing the default range from zero to a higher-than-maximum value.

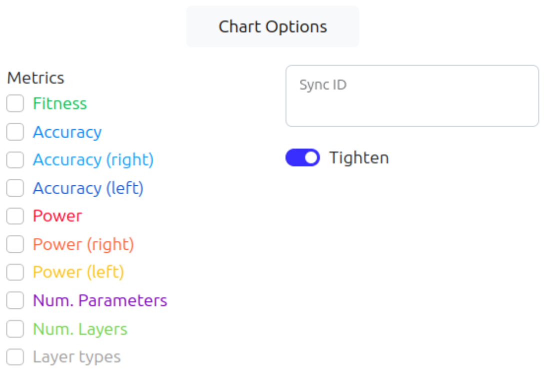
Individual inspection

When analyzing the results from NE experiments, it is essential to analyze particular individuals due to the metrics they show. To fulfill this, we allow inspecting one individual, i.e., a point in the chart. To do this, it is required to have only one run selected in the data controls. With this requirement satisfied, clicking on the desired point opens a modal window with its information (Figure 5.6).

It shows information on the run and generation of the individual, metrics such as accuracy, training time, number of epochs, number of trainable parameters, and



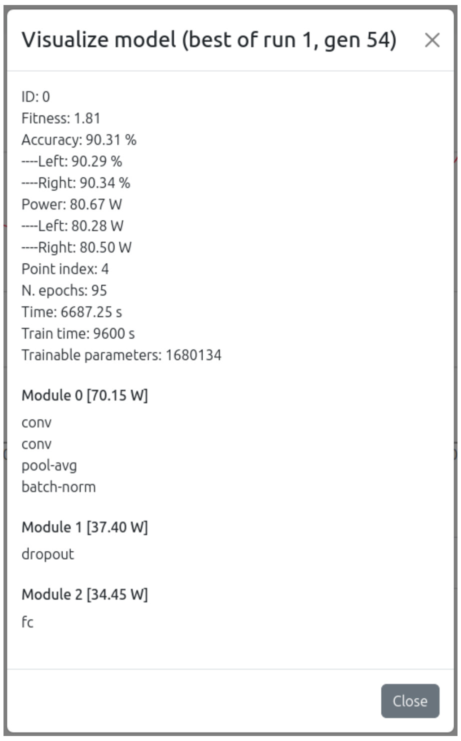
(a) Data options



(b) Chart options

Figure 5.5: Example visualization of the user's control on Neuroview.

power consumption. A list of the individual's modules of layers is also shown with the enumeration of the module's types of layers. The power consumption of the module is also presented.


Visualize model (best of run 1, gen 54)

ID: 0
Fitness: 1.81
Accuracy: 90.31 %
----Left: 90.29 %
----Right: 90.34 %
Power: 80.67 W
----Left: 80.28 W
----Right: 80.50 W
Point index: 4
N. epochs: 95
Time: 6687.25 s
Train time: 9600 s
Trainable parameters: 1680134

Module 0 [70.15 W]
conv
conv
pool-avg
batch-norm

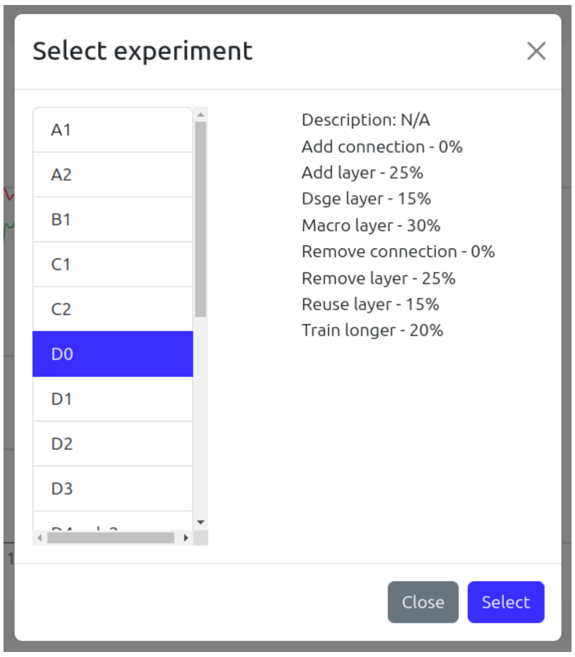
Module 1 [37.40 W]
dropout

Module 2 [34.45 W]
fc

Figure 5.6: Example inspection of an individual on NeuroView.

Experiment selection

An experiment can be selected through a modal window that lists all available experiments and the configurations of the one selected, such as the evolutionary parameters (Figure 5.7). This modal window is used in both *Select experiment* and *Compare with*. As expected, selecting an experiment affects the overall page through the changes on the charts and the data controls information.



Description: N/A
Add connection - 0%
Add layer - 25%
Dsge layer - 15%
Macro layer - 30%
Remove connection - 0%
Remove layer - 25%
Reuse layer - 15%
Train longer - 20%

Figure 5.7: Example selection of an experiment on NeuroView.

Experiment comparison

After selecting an experiment to be compared with the currently selected one, some minor but essential changes are performed. The data controls block changes its behavior to show information and settings about both experiments, and each chart controls stop allowing the selection of more than one metric.

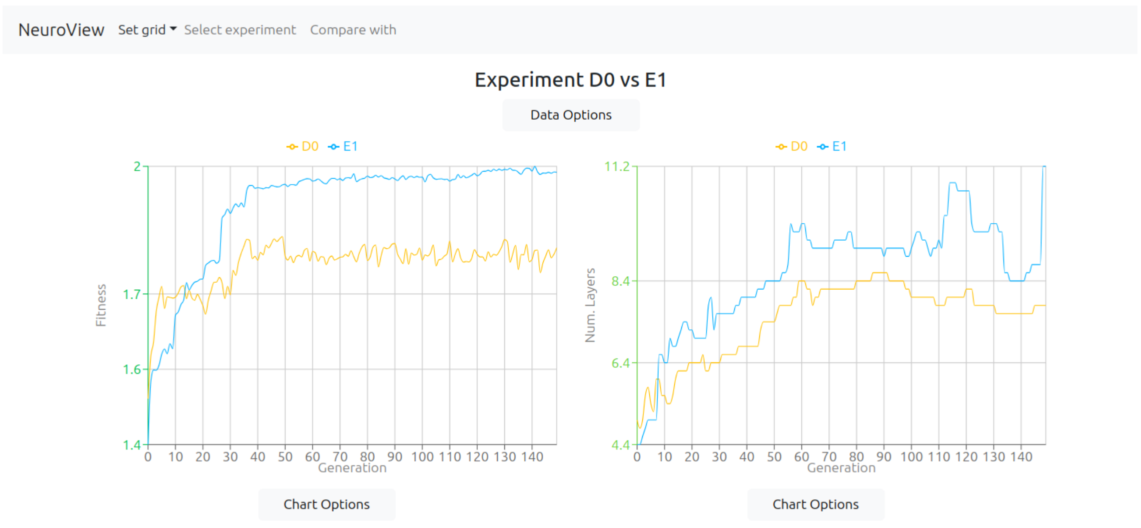


Figure 5.8: Example visualization of comparison of experiments on NeuroView.

Backend

The backend was developed in Python 3.9.13 with the Flask 2.3.1 framework at its center.

For deployment purposes, the backend server requires the type of environment it will execute on, i.e., ’development’ or ’production.’ This serves to indicate where the experimental settings and results are located.

The data files read by the NeuroView backend are exactly the JSON files that result from Fast-DENSER execution. They are files generated for each generation, and each one is essentially an array of individuals where every individual contains its phenotype, multiple measures over the epochs, and metrics such as accuracy, the number of trainable parameters, and power consumption. The settings files read by the backend are also in the JSON format specified bt Fast-DENSER.

The backend has two endpoints, one for obtaining all the experiments and their settings and another for getting an experiment’s results.

The first endpoint, */experiment/*, loads all the experiments’ settings JSON files, parses them, and inserts them into a hash map. Afterward, the map is sorted by the experiments’ titles, converted to JSON, and returned by the server.

The other endpoint, */experiment/<title>*, receives an argument, *<title>*, which is the title of the experiment the client wants. All files related to the solicited experiment are loaded and parsed, i.e., all generations from all runs available, and added into a map organized by the run number and the generation number. Each generation’s data is tidied to reduce the size of the HTTP response by discarding unnecessary metrics and measures. The server then returns all this data.

Frontend

The frontend was developed in Javascript within the React 18.2.0 framework ecosystem, i.e., using the framework and its packages through the *npm* package manager. As standard in React, it was designed with modularity and reactivity in mind, allowing for trivial structural modifications if required.

The Bootstrap CSS framework provides the overall design template and interface components, and they are used through the *react-bootstrap* package, a React wrapper for Bootstrap elements. The charts are powered by the *recharts* package, a chart library built within React with *D3.js*.

NeuroView consists of a single web page with the following sections: navigation bar, data controls, and $N \times M$ chart blocks, where the user defines N and M in the *Set grid* functionality. Each chart block consists of the chart itself and the chart controls.

Interactivity is obtained using the basilar *useState* and *useEffect* methods of React. The first one is responsible for managing the state within the React components, allowing for dynamic updates and re-rendering based on user interaction or data changes. The latter, *useEffect*, assists in handling side effects based on state updates or data changes. It also fetches data from the backend server on an async/await strategy.

As mentioned in 5.1.1, data manipulation is performed in the frontend to minimize loading time. To this matter, the *lodash* package is used. It is a versatile library with various utility functions that allow for simple and more efficient execution of tasks from data manipulation to array iteration.

5.1.3 Future development

An application of this type can continuously be enhanced based on user feedback and needs. In the future, Neuroview could benefit from an enhanced overall design (UI), an improved plan for user interactions and experience (UX), further testing to detect and address possible errors, and optimizations on the various Reactive states.

5.1.4 Summary

In this chapter, we introduced NeuroView, a tool to facilitate the interactive visualization of data originating from NE frameworks. NeuroView provides an intuitive and interactive interface to explore experiment results comprehensively. It allows the comparison of single and multiple metrics within and between experiments and a detailed examination of individual experiment data.

Its architecture comprises a Python backend built on Flask and a JavaScript frontend developed using React.

Chapter 6

Conclusion

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

The basilar approach measures the power consumed by the GPU on the inference phase of the DNN. We use the measure provided 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 a model tuned for environments with fewer resources, such as smartphones, while creating a mode power-intensive model tuned for environments with more resources, such as servers. This is performed in one training, thus taking less time to train the two models and saving energy. 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 favor the reintroduction of modules in a model according to the inverse of the power they consume, thus incorporating power-efficient modules into a model.

Furthermore, we developed a way to import previously trained models into Fast-DENSER and use them as the seed for the evolutionary process. This way, we expect to tune highly accurate models into models with lower power consumption.

Initially, two research questions were posed:

RQ1 — Can we train a single network that can be partitioned into a smaller, accurate, and efficient energy-wise partition? As verified in Chapter 4, we have validated the possibility of partitioning a model into two models by adding an additional output layer connected to an intermediate layer. We have also verified that the differences between the two partitions are marginal in accuracy and power usage. In the Power Experiment, we obtained smaller models, which are 0.4% less accurate than the full model whilst consuming less 1.49 W (2.1%). To obtain a more significant gap in power usage, it is likely necessary to increase the weight attributed to the power usage to allow the exploration of less accurate models that might show to be more power efficient.

RQ2 — To what degree does penalizing large energy consumption drive the search for efficient networks? We can drive the evolutions towards power-efficient models by using multi-objective fitness functions that penalize higher power consumption. Concretely, we obtained models that consume significantly less power than a baseline model whilst maintaining a similar or identical accuracy. This shows that it is possible to obtain efficient networks that are also accurate.

The results obtained by our proposals show that we can reduce the power consumption of the Artificial Neural Networks (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. In concrete, we can obtain models that consume less 19.5 W (19.5%) than a baseline model whilst having an accuracy 0.2% higher. The best model found regarding power consumes less 29.18 W (29.2%) 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 experiments where the mentioned strategies were applied. It could be valuable to explore other approaches and experiment more 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 application of only one strategy at a time. It would also be interesting to vary the fitness functions (e.g., the weight(s) used in them) and to vary evolutionary parameters such as the probabilities of the mutations.

Although we developed a process to start the evolutionary process with already trained models, a phenotype of the model is required in the format used by Fast-DENSER. To circumvent this or the manual creation of a phenotype of the model, it would be beneficial to enhance this process with the possibility of loading a model not originated in Fast-DENSER automatically. This would allow making well-established models more efficient power-wise.

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 [47, 48]. Such strategies would allow us to perform more experiments in less time, saving energy in the design process.

This page is intentionally left blank.

References

[1] D. Patterson, J. Gonzalez, U. Hölzle, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "The carbon footprint of machine learning training will plateau, then shrink," 2022.

[2] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *CoRR*, vol. abs/2006.05415, 2020.

[3] F. Assunção, N. Lourenço, B. Ribeiro, and P. Machado, "Fast-denser: Fast deep evolutionary network structured representation," *SoftwareX*, vol. 14, p. 100694, 2021.

[4] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," 2019.

[5] C. Darwin, *On the Origin of Species by Means of Natural Selection*. London: Murray, 1859. or the Preservation of Favored Races in the Struggle for Life.

[6] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 45, jul 2013.

[7] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd ed., 2015.

[8] P. A. Vikhar, "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.

[9] S. Luke, *Essentials of Metaheuristics*. Lulu, second ed., 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[10] A. P. Engelbrecht, *Computational Intelligence: An Introduction*. Wiley Publishing, 2nd ed., 2007.

[11] M.-J. Willis, H. Hiden, P. Marenbach, B. McKay, and G. Montague, "Genetic programming: an introduction and survey of applications," in *Second International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications*, pp. 314–319, 1997.

[12] R. McKay, N. Hoai, P. Whigham, Y. Shan, and M. O'Neill, "Grammar-based genetic programming: a survey," *Genetic Programming and Evolvable Machines*, vol. 11, pp. 365–396, 09 2010.

[13] C. Ryan, J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Genetic Programming* (W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, eds.), (Berlin, Heidelberg), pp. 83–96, Springer Berlin Heidelberg, 1998.

[14] C. Silva and B. Ribeiro, *Aprendizagem Computacional em Engenharia*. 01 2018.

[15] B. Yegnanarayana, *Artificial Neural Networks*. PHI Learning, 2009.

[16] H. D. Block, "The perceptron: A model for brain functioning. i," *Rev. Mod. Phys.*, vol. 34, pp. 123–135, Jan 1962.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[18] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022.

[19] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[20] M. c. Belgaid, R. Rouvoy, and L. Seinturier, "Pyjoules: Python library that measures python code snippets," Nov. 2019.

[21] R. Tang, W. Wang, Z. Tu, and J. Lin, "An experimental analysis of the power consumption of convolutional neural networks for keyword spotting," *CoRR*, vol. abs/1711.00333, 2017.

[22] E. Cai, D. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," *CoRR*, vol. abs/1710.05420, 2017.

[23] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pp. 477–484, 2016.

[24] Soumith, "Soumith/convnet-benchmarks: Easy benchmarking of all publicly accessible implementations of convnets." `https://github.com/soumith/convnet-benchmarks`, accessed on 25-Fev-2022.

[25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. `https://www.tensorflow.org/`, accessed on 25-Fev-2022.

[28] M. Lechner, R. M. Hasani, A. Amini, T. A. Henzinger, D. Rus, and R. Grosu, "Neural circuit policies enabling auditable autonomy," *Nature Machine Intelligence*, vol. 2, pp. 642–652, 2020.

[29] G. Cuccu, J. Togelius, and P. Cudré-Mauroux, "Playing atari with six neurons," *CoRR*, vol. abs/1806.01363, 2018.

[30] D. Xu, Y. Shi, I. W. Tsang, Y.-S. Ong, C. Gong, and X. Shen, "A survey on multi-output learning," 2019.

[31] S. Subedi, "Multi output neural network in Keras (Age, gender and race classification) — sanjayasubedi.com.np." `https://sanjayasubedi.com.np/deeplearning/multioutput-keras/` accessed on 11-Mar-2023].

[32] K.-H. Chang, "Chapter 5 - multiobjective optimization and advanced topics," in *Design Theory and Methods Using CAD/CAE* (K.-H. Chang, ed.), pp. 325–406, Boston: Academic Press, 2015.

[33] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "DENSER: deep evolutionary network structured representation," *CoRR*, vol. abs/1801.01563, 2018.

[34] C. Hsu, S. Chang, D. Juan, J. Pan, Y. Chen, W. Wei, and S. Chang, "MONAS: multi-objective neural architecture search using reinforcement learning," *CoRR*, vol. abs/1806.10332, 2018.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[36] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," 2018.

[37] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[38] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," 2019.

[39] Z. Lu, I. Whalen, V. Boddeti, Y. D. Dhebar, K. Deb, E. D. Goodman, and W. Banzhaf, "NSGA-NET: A multi-objective genetic algorithm for neural architecture search," *CoRR*, vol. abs/1810.03522, 2018.

[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, Apr. 2002.

[41] Y. Yusoff, M. S. Ngadiman, and A. M. Zain, "Overview of nsga-ii for optimizing machining process parameters," *Procedia Engineering*, vol. 15, pp. 3978–3983, 2011. CEIS 2011.

[42] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of bloom, a 176b parameter language model," 2022.

[43] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[44] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[45] M. Shi, Y. Zhou, Q. Ye, and J. Lv, “Personalized federated learning with hidden information on personalized prior,” 2022.

[46] M. S. Tanveer, M. U. K. Khan, and C.-M. Kyung, “Fine-tuning darts for image classification,” 2020.

[47] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, “Neural architecture search without training,” 2021.

[48] W. Chen, X. Gong, and Z. Wang, “Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective,” 2021.

This page is intentionally left blank.

Appendices

Appendix A

DENSER Grammar

```
<features> ::= <convolution> | <convolution> | <pooling> | <pooling> |
               <dropout> | <batch-norm>
<convolution> ::= layer:conv [num-filters,int,1,32,256]
                  [filter-shape,int,1,2,5] [stride,int,1,1,3]
                  <padding> <activation-function> <bias>
<batch-norm> ::= layer:batch-norm
<pooling> ::= <pool-type> [kernel-size,int,1,2,5] [stride,int,1,1,3]
              <padding>
<pool-type> ::= layer:pool-avg | layer:pool-max
<padding> ::= padding:same | padding:valid
<dropout> ::= layer:dropout [rate,float,1,0,0.7]
<classification> ::= <fully-connected> | <dropout>
<fully-connected> ::= layer:fc <activation-function>
                      [num-units,int,1,128,2048] <bias>
<activation-function> ::= act:linear | act:relu | act:sigmoid
<bias> ::= bias:True | bias:False
<softmax> ::= layer:fc act:softmax num-units:10 bias:True
<learning> ::= <gradient-descent> <early-stop> [batch_size,int,1,50,500]
               epochs:10000 | <rmsprop> <early-stop> [batch_size,int,1,50,500]
               epochs:10000 | <adam> <early-stop> [batch_size,int,1,50,500]
               epochs:10000
<gradient-descent> ::= learning:gradient-descent [lr,float,1,0.0001,0.1]
                       [momentum,float,1,0.68,0.99]
                       [decay,float,1,0.000001,0.001] <nesterov>
<nesterov> ::= nesterov:True | nesterov:False
<adam> ::= learning:adam [lr,float,1,0.0001,0.1] [beta1,float,1,0.5,1]
           [beta2,float,1,0.5,1] [decay,float,1,0.000001,0.001]
<amsgrad> ::= amsgrad:True | amsgrad:False
<rmsprop> ::= learning:rmsprop [lr,float,1,0.0001,0.1] [rho,float,1,0.5,1]
              [decay,float,1,0.000001,0.001]
<early-stop> ::= [early_stop,int,1,5,20]
<middle_point> ::= [middle_point,int,1,0,x]
```