Multi-objective Evolutionary Algorithms for the optimization of Deep Neural Network Architectures

By

Cosijopii García García

A dissertation submitted in partial fulfillment of the requirements for the degree of:

Doctor of Science in Computer Science

at

Instituto Nacional de Astrofísica, Óptica y Electrónica

February, 2025

Supervised by:

Dra. Alicia Morales Reyes

Dr. Hugo Jair Escalante Balderas

Multi-objective Evolutionary Algorithms for the optimization of Deep Neural Network Architectures

Computer Science Department

Doctoral thesis

By:

Cosijopii Garcia Garcia

Supervised by:

Dra. Alicia Morales Reyes

Dr. Hugo Jair Escalante Balderas

Instituto Nacional de Astrofísica Óptica y Electrónica

Tonantzintla, Puebla. February 2025

Abstract

Neural Architecture Search (NAS) has emerged as a critical area in deep learning, focusing on automating the design of convolutional neural networks (CNNs) to optimize their performance across different tasks. Despite advancements, most NAS approaches have predominantly focused on single-objective optimization, aiming primarily to maximize accuracy. This approach often overlooks other important factors, such as model complexity, computational time, and generalization capability. In this thesis, we address these gaps by introducing a multi-objective framework for NAS that leverages evolutionary algorithms (EAs).

Our contributions include the development of a novel search space representation for CNNs based on Cartesian genetic programming (CGP), designed to accommodate both architectural operations and hyperparameters flexibly. This representation enables a more efficient exploration of potential architectures, capturing a diverse range of high-performance models. Furthermore, we propose a progressive search strategy that incorporates self-supervised learning techniques to guide the evolutionary process more effectively. Additionally, a new performance estimation strategy is developed, based on incremental dataset expansion, to reduce computational costs during the search process.

To validate the proposed framework, extensive experiments were conducted on benchmark image classification tasks, comparing the performance of our multi-objective NAS approach with state-of-the-art NAS methods. The results demonstrate that our approach not only achieves competitive accuracy but also offers improved trade-offs between multiple objectives. The findings highlight the potential of evolutionary-based multi-objective optimization in advancing NAS methodologies, providing a pathway towards more effective and adaptable deep learning models for real-world applications.

Resumen

La búsqueda de arquitecturas neuronales (NAS) se ha convertido en un área crítica en el aprendizaje profundo; ésta se enfoca en automatizar el diseño de redes neuronales convolucionales (CNNs) para optimizar su rendimiento en diversas tareas. A pesar de los avances, la mayoría de los enfoques de NAS se han centrado predominantemente en la optimización de un solo objetivo, buscando principalmente maximizar la precisión. Este enfoque a menudo pasa por alto otros factores importantes, como la complejidad del modelo, el tiempo computacional y la capacidad de generalización. En esta tesis, abordamos estas limitaciones introduciendo un enfoque multiobjetivo para NAS que utiliza algoritmos evolutivos multiobjetivo (MOEAs). Nuestras contribuciones incluyen el desarrollo de una nueva representación del espacio de búsqueda para CNNs basada en la Programación Genética Cartesiana (CGP), diseñada para utilizar operaciones a nivel de capas como hiperparámetros; todo esto de manera flexible. Esta representación permite una exploración más eficiente de arquitecturas, capturando un amplio conjunto de modelos eficientes. Además, proponemos una estrategia de búsqueda progresiva que incorpora técnicas de aprendizaje autosupervisado para guiar el proceso evolutivo de manera más efectiva. Adicionalmente, se desarrolla una nueva estrategia de estimación de desempeño basada en la expansión incremental de conjuntos de datos, con el fin de reducir los costos computacionales durante el proceso de búsqueda.

Para validar el marco propuesto, se realizaron varios experimentos en tareas de clasificación de imágenes utilizando diferentes benchmarks, comparando el rendimiento de nuestro enfoque de NAS multiobjetivo con métodos NAS del estado del arte. Los resultados demuestran que nuestro enfoque no solo logra una precisión competitiva, sino que también ofrece mejores trade-offs entre múltiples objetivos. Los resultados obtenidos demuestran el potencial de la búsqueda basada en algoritmos evolutivos multiobjetivo para el diseño automatizado de arquitecturas neuronales, proporcionando soluciones que son tanto efectivas como eficientes en diferentes escenarios del mundo real.

Dedication

I feel compelled to dedicate this thesis to all those who have glimpsed something beyond what society considers normal, to those who refuse to conform to current demands, and who, on a higher level, strive to create something that benefits others. This dedication is also to life, to love, to friendship, to aptitude, and to the wisdom found in solitude.

Acknowledgments

I would like to express my deepest gratitude to my advisors, Dr. Alicia Morales Reyes and Dr. Hugo Jair Escalante Balderas, for their guidance and support throughout these years of hard work, their insights have been invaluable on this journey. I am also grateful to Dr. Bilel Derbel and Dr. Zakaria Abdelmoiz Dahi for their advice and mentorship during my time at INRIA. I would also like to express my sincere thanks to all those who, in one way or another, directly or indirectly, have contributed to the completion of this thesis.

My thanks go to INAOE for providing me with the necessary tools and facilities to conduct my research. I would also like to extend my appreciation to my reviewers: Dr. Alfonso Martínez Cruz, Dr. Jesús García Díaz, Dr. René A. Cumplido Parra, Dr. Saúl Zapotecas Martínez and Dr. Efrén Mezura Montes, for their constructive feedback and assistance in improving this work.

I am thankful to CONAHCYT for the financial support provided through scholarship No. 794300, and for the grant CB-S-26314 that supported this work.

The experiments presented in Chapter 6 were conducted using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, several universities, and other organizations (see `https://www.grid5000.fr`).

Finally i acknowledge the computer resources, technical advice, and support provided by the Laboratorio Nacional de Supercómputo del Sureste de México (LNS), a member of the CONAHCYT national laboratories, under project No. 202103083C.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Deep neural networks (DNNs), particularly convolutional and recurrent neural networks, have recently gained considerable popularity for approaching a wide variety of problems [2, 3, 4, 5]. Regarding convolutional neural networks (CNNs), these are very effective computational models that have been thoroughly investigated in a wide range of image processing and computer vision-related tasks. This has been possible thanks to a number of factors including availability of large amounts of data, high-performance computing resources and research advances in the machine learning field [6].

CNNs are often organized according to complex topologies (architectures) whose design, including hyperparameter configuration, is done by expert users. While CNNs grow in complexity, manual configuration becomes time consuming and in some cases unfeasible [6]. Neural Architecture Search (NAS) is a field that aims at automating the design and configuration of CNN models [1]. NAS methods explore the space of CNN topologies looking for an architecture that meets criteria, for instance, achieving a minimum performance, or being *light* in terms of the number of parameters.

Nowadays, progress in NAS research has resulted in the identification of novel CNNs with favorable performance on representative image classification datasets, attracting the scientific community's attention to this topic [7].

Evolutionary computation (EC) is a set of techniques inspired by the process of natural evolution to deal with complex optimization problems among others. The multi-objective approach deals with problems where objectives are in conflict and focuses on finding a set of solutions that represent a trade-off among them [8]. EC techniques have been successfully applied in the multi-objective optimization problems domain. In the context of NAS, the automated design of high-performance and low-complexity network architectures remains an open issue. This relevant problem can be formulated as one of multi-objective optimization [9, 7, 10, 11, 12].

NAS can be divided in three components: the search space, the search strategy, and the performance estimation strategy. Each of these components plays a crucial role in the overall design and performance of the neural architectures.

In this thesis, we have developed and refined each of these areas. The primary contribution is the design of a new search space that is both flexible and modular, capable of supporting hyperparameters. From an evolutionary computation perspective, a novel representation based on Cartesian Genetic Programming was created. Building on this representation, a new search strategy was proposed, leveraging self-supervised learning and progressive search techniques. Additionally, a new method for performance estimation of neural architectures was developed, aimed at reducing the evaluation time. The contributions of this thesis present a comprehensive set of strategies for designing neural architectures from a multi-objective perspective.

1.1 Problem statement

State-of-the-art CNNs are becoming increasingly complex, and their performance depends on both architecture and hyperparameter configuration, in addition to their high processing requirements. Furthermore, extensive research has been conducted through the development of specialized architectures for specific tasks [13]. However, due to the lack of understanding of DNNs, determining CNN performance without empirical benchmark testing remains challenging. Accurate and less complex architectures are desirable in scenarios where applications and user requirements needs models where time is a critical variable. However, searching for these optimal CNN architectures becomes a time-consuming black-box optimization task. Many research projects have been developed to automate CNN architectural search. Several of these works have posed this problem as a single-objective optimization problem, aiming for model accuracy on a specific dataset [14, 15, 16]. Recent works have reevaluated NAS and approached it as a multiobjective optimization problem, as more than one objective is involved in CNN development [10, 7, 17]. Important variables such as the number of parameters and architectural complexity, usually measured in MAdds(multiply-add operations), have a significant impact on the final model's performance and are strongly related to the application context. Recent research has focused on more classical Pareto-based algorithms. However, algorithms such as indicator-based MOEAs or decomposition remain unexplored.

1.2 Motivation

The recent and rapid advances in deep learning, particularly in CNNs, have highlighted the challenges and necessity of automating their design and configuration. The No Free Lunch (NFL) theorem [18, 19] suggests that no single optimization algorithm can be universally effective across all problems, which implies the need for specialized algorithms tailored to specific tasks. In the context of NAS, particularly NAS applied to CNNs, this specialization is crucial for achieving optimal performance on specific datasets and applications.

NAS has emerged as a vital technique for automatically designing neural networks. It leverages various search methodologies to discover architectures that balance performance and complexity. Traditionally, NAS approaches have focused on three primary paradigms: reinforcement learning (RL), evolutionary algorithms (EAs), and Bayesian optimization (BO) [1]. Each of these methods brings unique advantages and challenges, particularly in managing the trade-offs between multiple objectives, such as accuracy, latency, and model complexity.

This thesis focuses on evolutionary algorithms (EAs) due to their proven effectiveness in multi-objective optimization scenarios and their adaptability in representing solutions and designing operators specific to a problem's context. EAs offer flexibility in encoding architectures and dynamically adapting the search process, making them particularly suitable for the complex search spaces involved in NAS. Despite their potential, multi-objective evolutionary algorithms (MOEAs) have not been extensively explored in NAS, especially concerning CNNs, presenting a significant opportunity for further development.

Through the exploration of evolutionary approaches in NAS, this research aims to contribute to the understanding of how specialized algorithms can better navigate the trade-offs inherent in designing neural architectures. By leveraging EAs, the goal is to develop methods that can efficiently search for architectures that are both accurate and computationally efficient, providing a robust foundation for future innovations in deep learning.

1.3 Justification

NAS represents a rapidly evolving area within deep learning, with significant potential to automate the design of complex CNNs [1]. As deep learning models become increasingly complex, the challenge of manually designing optimal architectures has grown, necessitating more sophisticated automated methods. Despite substantial progress, most existing NAS approaches have primarily focused on single-objective optimization, typically aiming to maximize accuracy for specific datasets. This narrow focus limits the exploration of trade-offs between multiple critical objectives, such as model complexity, computational efficiency, and generalization ability.

Current research utilizing EAs in NAS has shown promise due to their flexibility and robustness in exploring large search spaces. However, studies from a multi-objective perspective often rely on binary or integer representations, overlooking other mechanisms that could be employed. This gap exists partly because the multi-objective approach is still under extensive investigation [9, 10, 7]. Such approaches may not fully exploit the potential of NAS, as they do not adequately address the multi-objective nature of real-world applications, where models must balance several performance metrics simultaneously.

A crucial aspect of NAS is the representation used to define the search space. The way neural architectures and their hyperparameters are encoded significantly influences the efficiency and effectiveness of the optimization process. An optimal representation should be both flexible and comprehensive, enabling the exploration of diverse architectures that achieve high performance across multiple objectives. This flexibility is particularly important given the dynamic nature of deep learning tasks and the need for models to adapt to different environments and data distributions.

To address these challenges, this thesis proposes a novel multi-objective NAS framework that incorporates advanced evolutionary strategies and a more sophisticated representation mechanism based on Cartesian Genetic Programming. By focusing on multi-objective optimization, this research aims to explore the trade-offs between competing objectives such as accuracy and model complexity. Additionally, this work introduces innovative strategies for both the search process and performance estimation, which are designed to enhance search efficiency and reduce computational overhead.

The contributions of this thesis comprise a set of strategies for the automated design of neural architectures from a multi-objective perspective. By addressing the current limitations in the field, this research aims to advance the state of the art in NAS, providing

a foundation for developing more effective and efficient deep learning models that can better meet the diverse needs of real-world applications.

1.4 Research questions

The primary aim of this thesis is to advance the field of NAS by leveraging evolutionary algorithms and innovative representation mechanisms to optimize neural network architectures across multiple objectives. The following research questions guided the investigation.

- Do different abstraction levels for solutions representation and population dynamics in multiobjective evolutionary algorithms positively affect Neural Architecture Search for significantly less complex and highly accurate Convolutional Neural Networks?
- Can multiobjective evolutionary algorithms improve neural architecture search for significantly less complex and highly accurate convolutional neural networks?
- What kind of multiobjective evolutionary paradigms can lead to less complex and highly efficient CNN architectures?
- What modifications to the search strategy will allow us to better exploit the search space?
- Will combining different strategies for performance estimation have a positive effect on the total search time?

1.5 Hypothesis

A novel solution representation in evolutionary algorithms, which accurately models convolutional neural network components such as layer operations and hyperparameters, when combined with optimized search operations and dynamic population management within multi-objective evolutionary algorithms, will enhance neural architecture search performance, particularly by improving accuracy and reducing computational cost (measured in multiply-add operations).

1.6 General objective

To design, develop, and evaluate a novel search space representation for convolutional neural networks, integrating specific layer operations and hyperparameters within a multi-objective evolutionary algorithm framework in order to enhance the neural architecture search process, optimizing for image classification tasks to achieve competitive performance in terms of accuracy and computational efficiency, specifically measured by Multiply-Add operations, compared to state-of-the-art methods.

1.6.1 Specific objectives

- To design a novel search space representation for convolutional neural networks (CNNs) that includes a comprehensive set of elements such as layer operations and hyperparameters, enabling more effective exploration of potential architectures.
- To develop and optimize evolutionary operators that are specifically tailored to the proposed search space representation, enhancing the search capabilities of multi-objective evolutionary algorithms in finding high-performing CNN architectures.
- To implement a new neural architecture search (NAS) algorithm utilizing the multi-objective evolutionary framework, to automatically discover CNN architectures that balance multiple objectives, particularly accuracy, and Multiply-Add operations
- To evaluate the proposed NAS algorithm's effectiveness by benchmarking its performance against state-of-the-art NAS methods, focusing on key metrics such as classification accuracy and Multiply-Add operations.

1.7 Contributions

The main contributions of this thesis are as follows:

- Design of a novel search space: The thesis introduces a flexible and modular search space for Neural Architecture Search that can effectively encode both architectural elements and hyperparameters of Convolutional Neural Networks. This new search space is based on Cartesian Genetic Programming, providing a robust foundation for the multi-objective evolutionary exploration of neural architectures.

- Development of an advanced search strategy: A novel search strategy is proposed, leveraging progressive self-supervised learning and a multi-objective evolutionary algorithm. This strategy is designed to improve the efficiency and effectiveness of the NAS process, enabling the discovery of *generic* architectures without requiring labeled data that balance accuracy and computational complexity.

- Design and implementation of a performance estimation strategy: this thesis also presents a new method for estimating the performance of neural architectures during the search process. This method dynamically adjusts the size of the dataset used for evaluation, starting with a small subset and gradually increasing it, thereby reducing the computational cost while maintaining the quality of the search outcomes.

The papers derived from this thesis are listed below:

- **Journals**

  - **C. Garcia-Garcia**, A. Morales-Reyes, and H. J. Escalante, **"Continuous Cartesian Genetic Programming based representation for multi-objective neural architecture search"**, Appl. Soft Comput., vol. 147, p. 110788, 2023.

- **International conferences**

  - **C. Garcia-Garcia**, H. J. Escalante, and A. Morales-Reyes, **"CGP-NAS: Real-based solutions encoding for multi-objective evolutionary neural architecture search"**, in Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA, 2022, vol. 1.

  - **C. Garcia-Garcia**, A. Morales-Reyes, and H. J. Escalante, **"Progressive Self-supervised Multi-objective NAS for Image Classification"**. In: Smith, S., Correia, J., Cintrano, C. (eds) Applications of Evolutionary Computation (Part of EvoStar). EvoApplications 2024. Lecture Notes in Computer Science, vol 14635. Springer, Cham. 3-5 April, Aberystwyth, Wales, United Kingdom.

  - **C. Garcia-Garcia**, B. Derbel, A. Morales-Reyes, and H.J. Escalante, **"Speeding up the Multi-Objective NAS Through Incremental Learning"**. 23rd

Mexican International Conference on Artificial Intelligence. Puebla. Mexico, 2024.

- **International workshops**
  - **C. Garcia-Garcia**, H. J. Escalante, and A. Morales-Reyes, **"Continuous Cartesian Genetic Programming based representation for Multi-Objective Neural Architecture Search for image classification"** (Extend abstract, poster presentation only) *LatinX in Computer Vision (LXCV) Workshop at CVPR, jun 18,2023, Vancouver, Canada*

1.8 Thesis summary

The remainder of this document is organized as follows:

- Chapter 2. Provides the theoretical foundations of NAS, evolutionary algorithms, and multi-objective optimization, with the aim of understanding the various concepts used in this thesis.
- Chapter 3. Presents a state-of-the-art analysis, divided into two parts: a single-objective and a multi-objective approach, focusing on NAS proposals from the perspective of evolutionary computation.
- Chapter 4. Introduces a novel search space based on a continuous representation using Cartesian Genetic Programming.
- Chapter 5. Describes a new search strategy based on progressive self-supervised learning, which is constructed by extending the proposed search space.
- Chapter 6. Presents an innovative performance estimation strategy based on incremental dataset expansion, building on the extended search space proposed earlier.
- Chapter 7. Recaps the research conducted, discusses future work, and provides the conclusions drawn from the results obtained.

Chapter 2
Background

This chapter provides an overview of the main topics involved in this research. It starts by presenting the fundamentals of neural architecture search, followed by discussions on deep neural networks, convolutional neural networks, and the classification problem. Finally, it delves into algorithmic techniques from the evolutionary computation area, considering its primary operations. Subsequently, an introduction to multiobjective optimization, along with the principal associated multiobjective evolutionary algorithms, is described.

2.1 Neural architecture search

In the last decade, the use of deep learning has become popular due to its high performance in learning tasks as well as its wide range of applications. It is known that the success of these applications is due to the architectures used, as well as the learning algorithms that optimize the associated weights. Therefore, the architectures used have become more complex, hence the need to automate the process of designing new architectures as well as the need for architectures for specific problems [1, 20, 21]. Neural architecture search (NAS) arises from this issue. This area can be divided into three different concepts: search space, search strategy, and performance estimation strategy

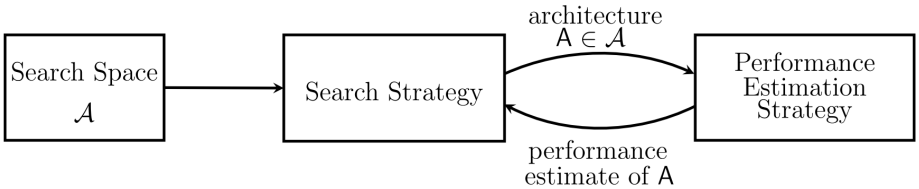


Figure 2.1: Neural architecture search, principal components, figure taken from [1].

2.1.1 Search space

As mentioned earlier, NAS can be divided into three main components, one of which is the search space. This is also one of the most important components because it defines how the architecture will be represented. This search space is encoded, and determines the complexity of the problem. Smaller search spaces can simplify the search process, whereas larger search spaces can provide more specific solutions [1]. Next, we will present two of the most commonly used methods in different approaches for NAS. The most basic and straightforward search space used in NAS is the *chain-structured neural network*. This search space consists of a sequence of n-layers. The size of this search space can be fixed, and the complexity of each layer depends on the types of operations chosen. Additionally, this search space can be either *linear* or allowing for more complex configurations such as *multi-branch networks* like the InceptionNet. In Figure 2.2, we can observe examples of both variants of the aforementioned search space [1, 22, 11].

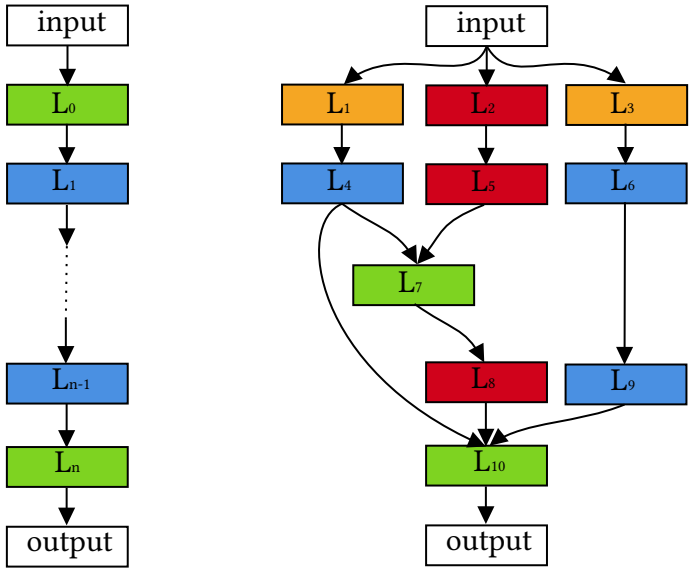


Figure 2.2: Each node represents a neural network layer, with different colors for different types. An edge from layer L_i to layer L_j indicates that L_j receives the output from L_i. The left side shows a chain-structured space, while the right side depicts a more complex space with additional layer types, branches, and skip connections.

One of the most commonly used search spaces is the *cell-based*, motivated by architectures where repetitive patterns were observed. Consequently, the search began to focus on these sets of blocks instead of the entire architecture. Typically, there are two types of cells: normal cells and reduction cells. Normal cells maintain the spatial size of the feature

maps, while reduction cells reduce the spatial dimension. Figure 2.3 illustrates an example of this search space [22, 1].

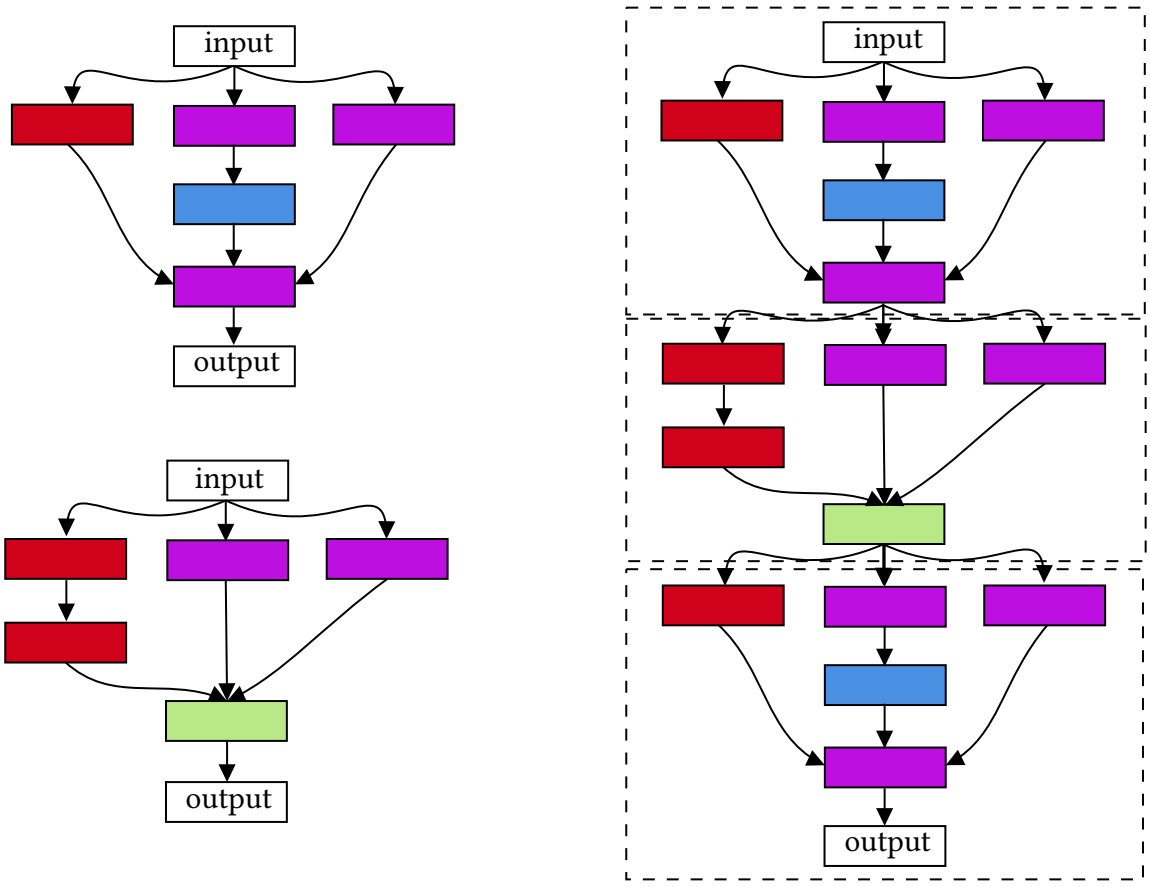


Figure 2.3: On the left, there are two types of cells: (top) a normal cell and (bottom) a reduction cell. On the right, we can observe the typical assembly used in this search space, where normal and reduction cells are usually interspersed; every color represents different layer operations.

A third type of search space emerges with the use of *cell-based* approaches. This method can be divided into two different search spaces: *macro-architecture* and *micro-architecture*. The macro architecture can be configured in various ways, including the connections between cells and the repetition of each block. Internally, the micro architecture search space defines which blocks or operations are specified in each layer and how they are interconnected. The choice of search space significantly influences the complexity of the optimization problem. For instance, even when dealing with just a single cell and focusing on the micro architecture, the problem remains complex for several reasons. These include the high dimensionality of the problem, as more intricate models typically

yield better performance [1]. The cell-based approach served as the based for our representation in this thesis.

2.1.2 Search strategy

The methods used in the search strategies vary widely in the state of the art and often depend on the type of search space. In this section, we will outline the most commonly used methods for NAS. Historically, evolutionary algorithms have played a significant role in evolving neural architectures and even their weights, with methods like NeuroEvolution of Augmenting Topologies (NEAT) [23]. However, with contemporary neural architectures containing millions of weights, it is now preferable to use methods based on Stochastic Gradient Descent (SGD). Evolutionary methods are primarily used to optimize the architecture, while gradient-based methods are used to optimize the weights [1].

Another widely used method is Bayesian optimization, which has been impactful since 2013 and remains a favorite in the machine learning community. Reinforcement learning has also been applied, adapting the search space into an action space where the agent's reward is based on estimating the performance of the trained architectures [1]. There are variants of the methods mentioned above, the most commonly used ones are presented here, and in Chapter 3 the state of the art of the different methods used in NAS will be review.

2.1.3 Performance estimation strategy

To guide the search process, it is necessary to estimate the performance of each solution. Traditionally, this is done using the training and validation sets to verify the quality of each solution. However, this method is very costly, requiring hundreds of GPU - days to complete the search. This highlights the need for new and faster methods to evaluate neural architectures [1].

With this issue in mind, multiple strategies have been developed, often aligned with different search spaces and search strategies. Table 2.1 presents four types of methods designed to accelerate performance estimation, addressing the challenge of time consumption. In this thesis we utilize lower fidelity estimate methods. These methods focus on using fewer epochs of training or smaller subsets of data to estimate the solution's performance while keeping the search time relatively short.

Table 2.1: Different strategies for estimating the performance of neural architectures as well as for accelerating this process [1].

Speed-up method	How are speed-ups achieved?
Lower fidelity estimates	Reduced training time by training for fewer epochs, on a subset of data, using smaller models, or downscaled data
Learning curve extrapolation	Shortened training time by extrapolating performance after just a few epochs
Weight Inheritance/ Network Morphisms	Models are warm-started by inheriting weights from a parent model instead of being trained from scratch
One-Shot Models/ Weight Sharing	Only the one-shot model needs training; its weights are shared across various architectures that are subgraphs of the one-shot model

2.2 Evolutionary computation

Evolutionary computing is a paradigm for solving problems; it is inspired by the biological principles of evolution. Among the most popular of which are genetic algorithms, evolutionary strategies, and genetic programming [8, 24].

In general, Evolutionary algorithms (EAs) are a class of optimization techniques inspired by the principles of natural selection and biological evolution. EAs are used to solve complex optimization problems by iteratively improving a population of candidate solutions. These algorithms are particularly well-suited for problems where the search space is large and complex. They have been successfully applied across various domains, including engineering, economics, artificial intelligence, and machine learning [8]. EAs consist of several essential components that work together to evolve a population of candidate solutions toward optimal or near-optimal solutions. The primary components of an EA include the population, representation (encoding), fitness function, selection, genetic operators, replacement, and termination criteria [25, 8]. In Figure 2.4 we can observe the principal components.

The population is a collection of candidate solutions, also referred to as individu-

als or chromosomes. Each individual represents a potential solution to the optimization problem. The population evolves over time, with each generation producing a new set of individuals through evolutionary operations. The size of the population can significantly impact the diversity of solutions and the convergence rate of the algorithm. On the other hand the representation, or encoding, defines how candidate solutions are structured within the algorithm. Common representations include binary strings, real-valued vectors, and more complex structures like trees or graphs. The choice of representation is crucial, as it directly impacts the performance of the evolutionary algorithm. For instance, in NAS, a flexible and modular representation that captures various aspects of neural networks, such as layer types and hyperparameters, is essential for effectively exploring the space of possible architectures.

The fitness function is a core component that evaluates how well each individual in the population solves the optimization problem. It assigns a fitness score to each individual, reflecting its quality or suitability as a solution. In a MOEA, multiple fitness functions may be used to evaluate different objectives simultaneously. For example, in NAS, objectives might include maximizing accuracy while minimizing computational cost. Having the fitness of each solution the selection process or parent selection determines which individuals from the current population will be chosen to create offspring for the next generation. Selection is typically based on the fitness of individuals, favoring those with higher fitness scores. Common selection methods include roulette wheel selection, tournament selection, and rank-based selection. The goal of selection is to ensure that better solutions have a higher chance of contributing to the next generation while maintaining diversity within the population.

Genetic operators such as crossover (recombination) and mutation are applied to individuals in the population to generate new offspring. Crossover combines two or more parent solutions to produce one or more offspring, promoting the exchange of genetic material and potentially creating better solutions. Mutation introduces small random changes to an individual's structure, maintaining diversity within the population and preventing premature convergence to suboptimal solutions. The design and application of these operators are critical for the algorithm's ability to explore and exploit the search space effectively.

Replacement strategies or survivor selection strategies define how the new offspring generated through genetic operators are integrated into the population. This strategy determines which individuals are retained for the next generation and which are discarded.

Replacement strategies can vary, from replacing the entire population with new offspring to more conservative approaches where only a portion of the population is replaced, preserving some of the best individuals from the current generation.

Lastly, termination criteria specify when the evolutionary algorithm should stop running. Common termination conditions include reaching a maximum number of generations, achieving a satisfactory fitness level, or observing no significant improvement in fitness over a specified number of generations. The choice of termination criteria can affect the balance between exploration and exploitation in the search process.

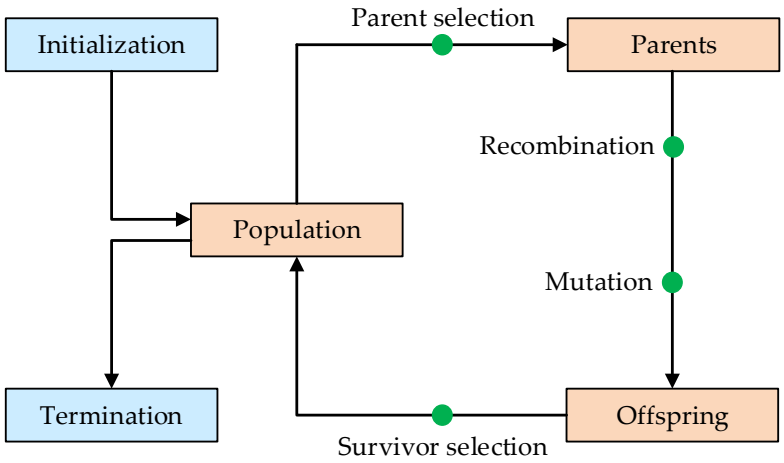


Figure 2.4: Evolutionary algorithm, general scheme.

2.2.1 Genetic programming

Genetic Programming (GP) is an evolutionary algorithm-based methodology inspired by natural selection to automatically evolve computer programs or models to solve specific problems. Introduced by John Koza in the 1990s, GP extends the principles of genetic algorithms (GAs) by evolving programs represented as trees or graphs [26]. The process starts with generating an initial population of programs, selecting the fittest individuals, and applying genetic operators such as crossover and mutation to produce new generations of programs.

In GP, programs are typically encoded as tree structures where nodes represent functions and leaves represent variables or constants, its commonly have a function set with this functions and constants, normally the function have a defined arity. The fitness of

each program is evaluated based on its performance on a given task. Through iterative evolution, GP seeks to discover solutions that perform well according to predefined criteria.

GP has been successfully applied to various domains, including symbolic regression, automated design, and optimization problems. Its flexibility in representing complex solutions and adaptability to different problem spaces make it a powerful tool for evolving models and algorithms [26].

2.2.2 Cartesian genetic programming

Cartesian genetic programming (CGP) was initially conceived to evolve digital circuits. It was proposed by Miller in 1997 [27] as a general form of genetic programming (GP) in 2000 and is called Cartesian because it represents a program using a two-dimensional grid [28]. CGP, unlike Genetic Programming (GP), is based on acyclic graphs for solution representation that allows forwarding connections. CGP shares important features with GP, such as the definition of a set of functions and their arity. Figure 2.5 shows a CGP solution representation example in which the mapping of the solution to its phenotype, in this case, a CNN architecture, is shown. However, CGP can also be applied to different areas, such as automatic design of digital circuits, mathematical equations, and even artistic applications [28].

In the genotype space, a solution is mapped as an integer-based vector divided by segments that represent the function identifier (this refers to a function predefined on a set of functions that are taken as nodes) and its connections. The size of each segment varies depending on the maximum arity of the represented function; in the example, the maximum arity is two. Another CGP characteristic is the inactive nodes that are not expressed in the phenotype (in Figure 2.5, these are represented by the grey sections), but it is information that is maintained in the genotype and is exploited during the evolutionary process, for example, by changing an important connection between two functions that maps as a big change in the phenotype. Considering this CGP scenario, the crossover operator is not used (but can be implemented). Instead, only mutations occur as random changes in connections and nodes.

The CGP is set up on a fixed-size grid $N_r \times N_c$, with the number of connections between nodes determined by an l variable known as level-back. Normally, the number of inputs and outputs depends on the problem. CGP in its base version is combined with the (1 +

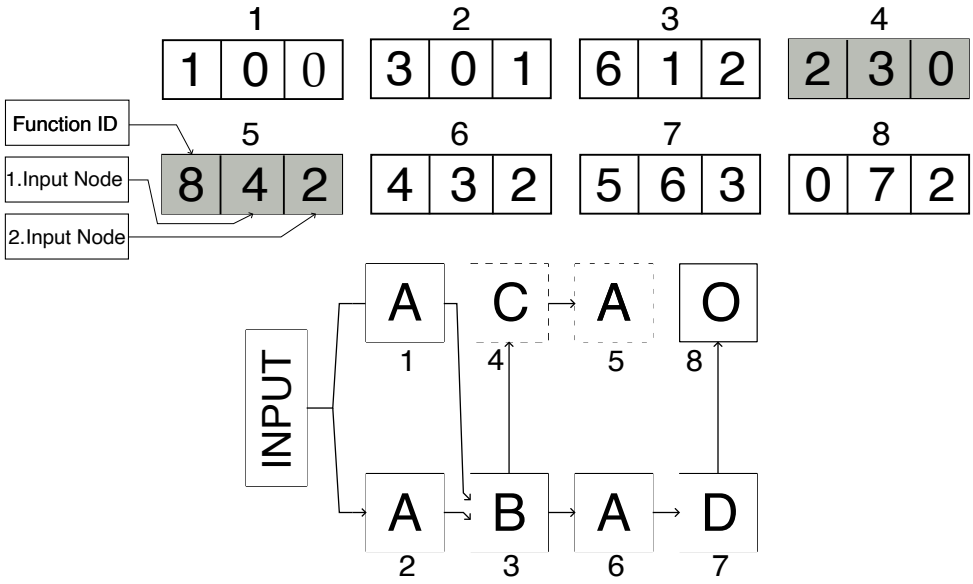


Figure 2.5: CGP representation. Top: integer-based genotype representation, Bottom: mapping to the phenotype space as an acyclic graph. Vectors in gray represent no connection, thus these are inactive nodes in the acyclic graph [29].

λ) evolutionary strategy algorithm. However, CGP can be adapted to other evolutionary searching algorithms.

2.2.3 Multiobjective optimization

In multiobjective problems (MOPs), solution quality is determined by the relationship between several conflicting objectives [8, 30]. Solving MOPs involves identifying trade-offs among all objective functions. Unlike single-objective problems, MOPs yield a set of optimal solutions instead of a single one. This is because it's impossible to find a single solution optimizing all objective functions simultaneously in multiobjective optimization. There are various approaches to address this issue. For instance, the weight sum method combines the fitness of each function to obtain a single measure, typically by assigning fixed weights to each function [8, 31, 30]. Another classic method is the ϵ-Constraint method, which considers one function as the main objective and treats the others as constraints [30, 31]. Goal programming aims to find solutions close to predefined objectives for each target and adjusts these objectives if the desired solutions are not achieved [30]. Multiobjective evolutionary algorithms surpass in obtaining a set of Pareto-optimal solutions without requiring prior knowledge of the problem, such as weight vectors. However,

classic methods may struggle to find Pareto optimal solutions in non-convex MOPs due to the complexity of the Pareto front and the difficulty in locating such solutions [30].

Basic concepts

This section explains basic concepts of multiobjective optimization.

- **Decision variables** are represented by a vector $\mathbf{x}$ with n decision variables represented by Equation 2.2.1 [31].

$$\mathbf{x} = [x_1, x_2, \ldots, \mathbf{x}_n] \tag{2.2.1}$$

- **Constraints** are imposed by environment characteristics or resources and occur in most optimization problems. They are expressed in the form of mathematical equalities or inequalities, represented in Equations 2.2.2 and 2.2.3. If the number of equality constraints is greater than the number of decision variables, the problem is over constrained so there are not enough degrees of freedom for optimization [31].

$$h_j = 0 \ , \ j = 1, 2, \ldots, p \tag{2.2.2}$$

$$g_i \leq 0 \ , \ i = 1, 2, \ldots, m \tag{2.2.3}$$

- **Objective function,** in multiobjective optimization, a set of objective functions are used to evaluate the decision variables vector: $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})$ where m is the number of objective functions in a multiobjective problem. The vector of objective functions $\mathbf{f}(\mathbf{x})$ is defined as [31]:

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})] \tag{2.2.4}$$

- **Decision variable and objective function space** are defined by an n-dimensional space, where each coordinate axis represents a component of a decision variable vector $\mathbf{x}$. The objective function space is defined by a m-dimensional space, with each coordinate axis corresponding to a component of the vector $\mathbf{f}_m(\mathbf{x})$. Each point in the decision variable space represents a solution, and when this vector is evaluated in the objective function, the resulting value is a point in the objective function space, determining the solution's quality. Thus, a function $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ maps the

decision variable space to the objective function space [31]. Figure 2.6 illustrates this process.

Multiobjective optimization problem

A multiobjective optimization problem involves multiple conflicting objective functions, which must be minimized or maximized simultaneously. Those functions could be subject to a number of constraints and variable bounds. Mathematically, a Multiobjective Optimization Problem (MOP) is defined as:

Definition 2.2.1. General MOP [32]:

$$\left.\begin{aligned} \text{Minimize/Maximize } \mathbf{f}_m(\mathbf{x}), m = 1, 2, \ldots, k; \\ \text{subject to } g_j(\mathbf{x}) \geq 0, j = 1, 2, \ldots, m; \\ h_k(\mathbf{x}) = 0, k = 1, 2, \ldots, p; \\ x_i^{(L)} \leq x_i \leq x_i^{(U)}, i = 1, 2, \ldots, t; \end{aligned}\right\} \tag{2.2.5}$$

with k objectives, m and p are the number of inequality and equality constraints. A solution $\mathbf{x} \in \mathbf{R}^n$ is a vector of n decision variables: $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, which satisfy all constraints and variable bounds [32, 31, 30]. The last set of constraints is called variable bounds, which restrict each upper and lower decision variable x_i value. These limits are the decision variable space size. If a solution $\mathbf{x}$ satisfies all restrictions and variable bounds, it is known as a ***feasible solution***. The set of all feasible solutions is called the ***feasible region*** [30]. It can be seen in Figure 2.6 that solutions within the blue area are feasible solutions, and their set determines a feasible area.

Dominance and Pareto optimality

In multiobjective optimization problems, several objectives conflict, this means that more than one optimal solution exists. These solutions are known as *Pareto-optimal* solutions. Definition of a Pareto-optimal solution is related to the domination concept as follows:

Definition 2.2.2. Pareto dominance [31]: A vector $\mathbf{u} = (u_1, u_2, \ldots, u_k)$ is said to dominate another vector $\mathbf{v} = (v_1, v_2, \ldots, v_k)$ (denoted by $\mathbf{u} \preceq \mathbf{v}$) if and only if $\mathbf{u}$ is partially less than $\mathbf{v}$, this is specified as follows: $\forall i \in \{1, \ldots k\}, u_i \leq v_i$ and $\exists i \in \{1, \ldots k\} : f(u_i) < f(v_i)$.

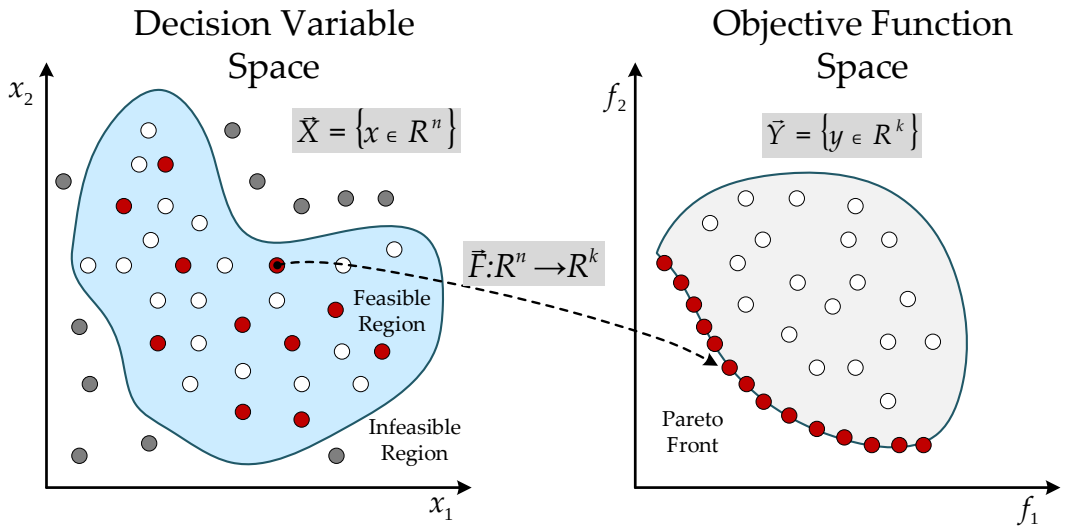


Figure 2.6: Decision variables space to objective function space mapping. Feasible solutions region is in blue. In the decision variable space, the Pareto optimal set is shown as red dots and its mapping to the objective function space creates the Pareto front.

Definition 2.2.3. Pareto Optimal Set [31], for a given MOP and $\mathbf{f}(\mathbf{x})$, the POS $\mathcal{P}^*$ is determined by:

$$\mathcal{P}^* = \{\mathbf{x} \in \Omega \mid \neg\exists \mathbf{x}' \in \Omega \, \mathbf{f}(\mathbf{x}') \preceq \mathbf{f}(\mathbf{x})\} \tag{2.2.6}$$

These solutions are represented in the decision variable space. Non-dominated solutions represented in the Pareto optimal set are the best solutions with a trade-off among objectives. When mapping these solutions to the objective function space, a set called Pareto front ($\mathcal{PF}^*$) is defined next [31]:

Definition 2.2.4. For a given MOP, $\mathbf{f}(\mathbf{x})$ and POS, $\mathcal{P}^*$, the Pareto Front $\mathcal{PF}^*$ can be expressed as:

$$\mathcal{PF}^* = \{\mathbf{u} = \mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\} \tag{2.2.7}$$

| In Figure 2.6, a mapping example of solutions from the Pareto optimal set to the objective function space is shown, therefore the Pareto front is created with solutions with a trade-off among objectives.

2.2.4 Special solutions

This section explains three types of special solutions frequently used in multiobjective optimization algorithms: ideal, utopian, and nadir objective vectors, as illustrated in Figure

2.7.

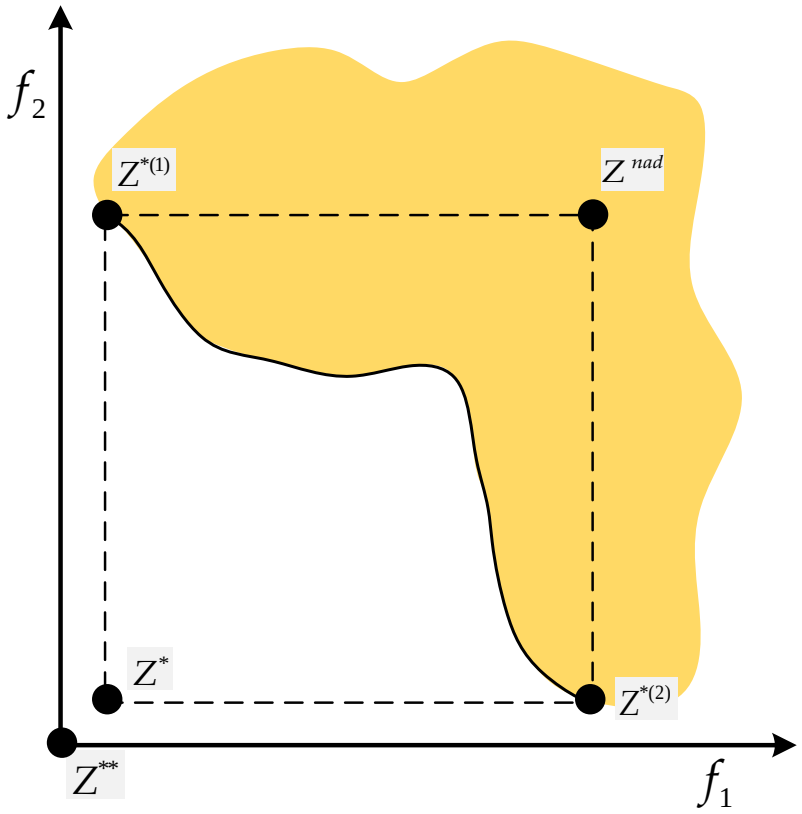


Figure 2.7: Representation of an objective function space with ideal (Z^*), utopian (Z^{**}), and nadir (Z^{nad}) objective vectors.

Ideal objective vector

Each conflicting objective has a unique optimal solution. The ideal objective vector consists of these optimal objective values [30].

Definition 2.2.5. The m-th component of the ideal objective vector Z^* is a constrained minimum solution of [30]:

$$\left.\begin{array}{l} \text{Minimize } \mathbf{f_m}(\mathbf{x}) \\ \text{subject to } \mathbf{x} \in S \end{array}\right\} \tag{2.2.8}$$

If the minimum solution for the m-th objective is the decision vector $x^{*(m)}$ with function value f_m^*, then the ideal vector is defined as [30]:

$$Z^* = \mathbf{f}^* = \left(f_1^*, f_2^*, \ldots, f_m^*\right) \tag{2.2.9}$$

where $\left(f_1^*, f_2^*, \ldots, f_m^*\right)$ are the solutions with the maximum value in the m objectives.

The ideal objective vector corresponds to a non-existent solution, since the solutions of Equation 2.2.9 for each objective function need not be the same; furthermore, many algorithms require knowledge of the lower bounds on each objective function to normalize objective values in a common range. In this case, the ideal vector is taken as the solution with the lowest value of solutions in the current population [30].

Utopian objective vector

The ideal objective vector consists of all objective functions at their lower bounds. This means that for each objective function, there exists at least one feasible solution in the solution space that shares an identical value with its corresponding elements in the ideal solution. However, some algorithms require a solution where the objective value is strictly better than any other solution in the search space. For this purpose, the utopian objective vector is defined [30]:

Definition 2.2.6. In a utopian objective vector Z^{**}, each component is slightly smaller than the ideal objective vector, or $Z_i^{**} = Z_i^* - \epsilon_i$ with $\epsilon > 0$ for all $i = 1, 2, \ldots, m$.

Like the ideal objective vector, the utopian objective vector also represents a non-existent solution [30].

Nadir objective vector

The nadir objective vector is the opposite of the ideal objective vector and represents the upper limit of each objective in the entire search space. It defines the upper limit of each objective vector in the Pareto optimal set, not throughout the search space [30]. In some algorithms, such as MOEA/D, the nadir point is represented as the solution which is the upper limit [33], the maximum of every f_m objective can be defined as:

$$Z^{nad} = \mathbf{f}^* = \left(f_1^*, f_2^*, \ldots, f_m^*\right) \tag{2.2.10}$$

where $\left(f_1^*, f_2^*, \ldots, f_m^*\right)$ are the solutions with the maximum value in the m objectives.

2.2.5 Multiobjective optimization evolutionary algorithms

The multiobjective evolutionary algorithms (MOEAs), in comparison with the classic mathematical programming methods mentioned briefly in the previous section, possess certain characteristics and advantages that make them well-suited for solving MOPs [30]. In this section, various evolutionary algorithm approaches for addressing multiobjective problems are reviewed. These algorithms can be categorized into three main paradigms: Pareto-based MOEAs, decomposition-based MOEAs, and indicator-based MOEAs [34].

Pareto-based paradigms

Pareto-based MOEAs use a dominance-based ranking scheme and combine elitist strategies that converge to a global optimum in some problems [8]. Pareto and elitist strategies laid the foundation for one of the most important algorithmic approaches in the area: the NSGA-II algorithm proposed by Deb et al. [35]. Pareto-based MOEAs commonly employ Pareto dominance, with some diversity criteria based on secondary ranking. Notable algorithms in this class include the Multiobjective Genetic Algorithm (MOGA) [36], the first MOEA; the Pareto Archived Evolutionary Strategy (PAES) [37], which uses a mesh in the objective function space to ensure all regions are visited; and the Strength Pareto Evolutionary Algorithm (SPEA) [38], which employs a different criterion based on dominance. SPEA ranks individuals by how many individuals they dominate and how many dominate them, and it also utilizes clustering. In its second version, SPEA-2 [39], both criteria are improved.

NSGA-II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) was proposed by Deb et al. in 2002 [35]. Its main features include the use of elitism, a diversity mechanism, and a focus on non-dominated solutions. This improved version of NSGA [40] aims to reduce the complexity of non-dominated sorting, make more efficient use of elitism, and reduce parameters.

NSGA-II begins with a random population, where each offspring is generated using two parents selected through a binary tournament method. The parents are recombined using the SBX operator and mutated via polynomial mutation [35]. The offspring set Q is then combined with the current population P, forming $P \cup Q$. The best μ individuals are selected from $P \cup Q$ based on the front to which they belong, determined by non-dominated

sorting (See Algorithm 2). This sorting ranks solutions by corresponding fronts, providing a quality measure to evaluate different solutions. If the set of μ solutions exceeds the population size, a ranking process called *crowding distance* (see Algorithm 1) is applied. Solutions with larger crowding distances are better ranked to maintain population diversity. Crowding distance is calculated as the average distance to neighboring solutions, forming a cuboid in the objective function space. Finally, μ solutions are passed on to the next generation, and the process repeats as described in Algorithm 3. Recently, NSGA-II has been adapted to address many objectives in its NSGA-III version [41], which combines NSGA-II principles with decomposition approaches.

Algorithm 1: Crowding Distance [35]

$l = |I|$;
$I[i]_{distance} = 0$;
for *each objective* m **do**
    $I = sort\ (I, m)$;
    $I[1]_{distance} = I[l]_{distance} = \infty$;
    $i = 2$ to
    $(l-1)$ I[i]$_{distance} = I[i]_{distance} + (I[i+1].m - I[i-1].m))\ /\left(f_m^{max} - f_m^{min}\right)$;

Decomposition-based paradigms

A common issue with MOPs involving more than three objectives is the ineffectiveness of dominance, rendering Pareto front ranking impractical. Consequently, decomposition-based methods have been incorporated into MOEAs. These methods can handle problems with two, three, or multiple objectives, offering a robust and effective alternative. One of the most significant algorithms in this category is MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition), developed by Zhang and Li [42, 8, 43].

MOEA/D

The Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D) integrates elements of the weighted-sum approach and population-based algorithms. MOEA/D begins by distributing a set of λ weight vectors in the objective function space and then creates an array of the T closest vectors using Euclidean distance, thereby forming neighborhoods. MOEA/D and its variants show performance comparable to Pareto-based algorithms for problems with a few objectives and superior performance for prob-

Algorithm 2: Non-dominated sorting [35]

```
for each p ∈ Population P do
    S_p = ∅, n_p = 0;
    for each q ∈ P do
        if p ≺ q then
            S_p = S_p ∪ {q};
        else
            if q ≺ p then
                n_p = n_p + 1;
        if n_p = 0 then
            p_rank = 1;
            F_1 = F_1 ∪ {p};
while F_i ≠ ∅ do
    Q = ∅;
    for each p ∈ F_i do
        for each q ∈ S_p do
            n_q = n_q − 1;
            if n_q = 0 then
                q_rank = i + 1;
                Q = Q ∪ {q};
    i = i + 1;
    F_i = Q;
```

Algorithm 3: NSGA-II [35]

```
Create initial population P_t;
Evaluate fitness of each solution;
Apply non-dominated sorting to rank the solutions;
while Termination condition not satisfied do
    Offspring population Q_t = ∅;
    for each solution in P_t do
        Select two parents using binary tournament;
        Recombine the parents using SBX and generate a child r;
        Apply mutation on r generating q;
        Q_t = Q_t ∪ q;
    Apply Algorithm 2 to rank P_t ∪ Q_t population obtaining F fronts;
    if |F_1| + |F_2|, ..., |F_i| = |P_t| then
        Copy the solutions of these fronts to the new population P_{t+1};
        P_{t+1} = ⋃_i F_i;
    else
        Determine the front H of F_i that is greater than |P_t|, to this last front
          apply Crowding Distance (Algorithm 1) and add to P_t the solutions with
          the higher Distance;
        P_{t+1} = (⋃_i F_i)) ∪ H;
```

lems with five or more objectives [8, 42]. The canonical MOEA/D algorithm employs Tchebycheff decomposition, defined as follows:

$$\min g^{te}\left(\mathbf{x}|\lambda^j, Z^*\right) = \max_{1\leq i\leq m} \frac{1}{\lambda_i^j}\left|f_i\left(\mathbf{x}\right) - Z_i^*\right| \tag{2.2.11}$$

The MOEA/D algorithm is detailed in Algorithm 4. Initially (lines 1-4), λ reference vectors are set up, neighborhoods are created using the nearest T vectors, and the ideal point Z is calculated. The main cycle iterates over all individuals in the population. In line 7, two parents are selected from the neighborhoods stored in $B(i)$. Two random parents are chosen from this structure, and the SBX operator along with polynomial mutation is used to generate the offspring. It is important to note that only one child is generated in this process. In the final steps of the algorithm, the value of Z is updated, and the aggregation values of the two parents and the offspring y^i are calculated using the λ reference vectors. Finally, if the offspring y^i has a smaller aggregation value than one of the parents, it replaces the parent; otherwise, the parent remains, and the population is not modified.

Algorithm 4: MOEA/D

1 $\lambda^i = (\lambda_1^i, \ldots, \lambda_m^i)^T, i = 1, \ldots, N_p$;
2 $B(i) = \{i_1, \ldots, i_t\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.;
3 $P = \{\mathbf{x}^1, \ldots, \mathbf{x}^{N_p}\}$;
4 Set Z using equation 2.2.9;
5 **while** $k \leq T_{max}$ **do**
6 **for** $i = 1$ **to** $size(P)$ **do**
7 $P = B\left(i, randperm(B)\right)$;
8 Generate y from $\mathbf{P(1)}$ and $\mathbf{P(2)}$ by GA operator;
9 Polynomial mutation on y to new solution y^i;
10 Update Z using equation 2.2.9;
11 $g_{pop} = g^{te}\left(P|\lambda^P, z^*\right)$;
12 $g_y = g^{te}\left(y^i|\lambda^P, z^*\right)$;
13 $Population\left(P\left(g_{pop} \geq g_y\right)\right) = y^i$;

Indicator-based paradigms

Indicator-based MOEAs utilize quality metrics, which are functions assessing approximation sets, to define selection mechanisms. The underlying concept aims to optimize the population's indicator value throughout the evolutionary process. A prominent exam-

ple of these algorithms is the S-Metric Selection Evolutionary Multi-objective Algorithm (SMS-EMOA) [44]. This algorithm employs the Hypervolume (HV) [45] indicator, which measures the dominated volume of an approximation set bounded by an anti-optimal point. SMS-EMOA establishes a total order among solutions by evaluating their contribution to the hypervolume indicator [46].

SMS-EMOA

The S-Metric Selection Evolutionary Multiobjective Optimization Algorithm (SMS-EMOA) is a variant of NSGA-II in which the crowding distance-based density estimator is replaced by the Hypervolume (HV) indicator. In other words, solutions that contribute the least to the HV are eliminated, as described in Algorithm 4. Due to the mathematical properties of HV, this algorithm can theoretically solve any MOP, producing a uniform and well-distributed Pareto front. However, one of the challenges with this algorithm is that when addressing Many Objectives Problems (MaOPs), it often requires calculating the HV for the entire population. This leads to additional computation and a high cost in calculating the HV in high-dimensional objective spaces.

Algorithm 5: SMS-EMOA

```
Randomly initialize population P of size μ;
while Termination condition not satisfied do
    Create a new offspring solution q;
    Q = P ⋃ q;
    F_1, ..., F_k =non-dominated sorting(Q) using Algorithm 2;
    if |F_K| > 1 then
        z_i^max = max_{q∈Q} fi(q), ∀_i = 1, ..., m;
        q_worst = arg min_{q∈R_k} HV (Q, z^max) − HV (Q\{q}, z^max)
    else
        q_worst is equal to the sole solution in R_k;
    P = Q\{q_worst}
```

Hypervolume

This metric (HV) reflects the closeness between $\mathcal{PF}^*$ and $\mathcal{PF}_{true}$. A large HV indicates that the $\mathcal{PF}^*$ set is closer to $\mathcal{PF}_{true}$. HV corresponds to the non-overlapping volume of all hypercubes formed by the reference point z and every vector in the $\mathcal{PF}^*$. An HV with

a larger value represents better performance in terms of both diversity and convergence. Let Λ denote the Lebesgue measure in $\mathbb{R}^m$,then HV is defined as follows:

$$HV(\mathcal{A}, z) = \Lambda\left(\bigcup_{a \in \mathcal{A}} \{x \mid a \prec x \prec z\}\right) \tag{2.2.12}$$

where $z \in \mathbb{R}^m$ is a reference point that should be dominated by all points in $\mathcal{A}$. The HV is a unary quality indicator (QI) that evaluates both the convergence and the overall spread of an approximation set, regardless of its dimensionality. It does this by measuring the volume of the objective space that is dominated by $\mathcal{A}$ At present, HV is the only unary QI that is proven to be Pareto compliant. [46].

2.3 Deep neural networks

Deep learning is a branch of machine learning, which falls under the domain of artificial intelligence. As the basic idea of machine learning is to learn from data, the representation of data is particularly important in the performance of learning algorithms. Obtaining the most effective features is not a simple task and often requires a certain degree of human expertise and prior knowledge of the task at hand. This makes the process somewhat dependent on human input.

Representation learning has emerged as an alternative to feature engineering, capable of automatically extracting useful representations. By using representation learning, raw data can be mapped to a useful representation that can be more easily utilized by a classifier or predictor. Deep learning learns a complex representation by aggregating simpler ones, which in turn depend on even simpler representations. By using this hierarchy of complex mappings, deep neural networks can learn complex concepts [47].

2.3.1 Neural networks

An artificial neuron mimics the behavior of a biological neuron, although a simplified model is used due to the complexity of accurately replicating the biological system. This model includes input signals from other neurons, a threshold function, and an output. The simple perceptron, introduced by Rosenblatt and Minsky [48, 49], is depicted in Figure 2.8. The perceptron can be described using a linear discrimination function (see Equation 2.3.1). As a linear classifier, a single perceptron can categorize patterns into two classes.

It takes a feature vector x as input and generates a scalar output [48].

$$y = \varphi\left(\sum_{i=1}^{n}(w_i x_i + w_0)\right) \tag{2.3.1}$$

Here, w represents the weights that define the hyperplane, and φ is a nonlinear function with a threshold, typically a sigmoid function [50].

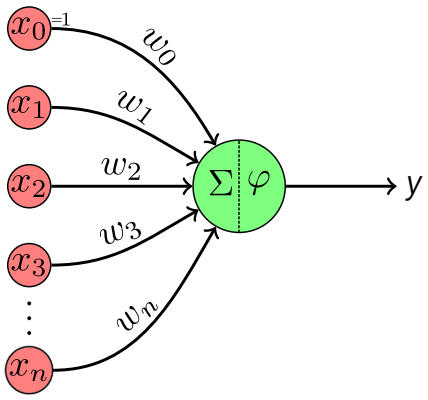


Figure 2.8: Representation of a simple perceptron.

Neural networks consist of nodes or units interconnected by links. Each link has an associated weight w that influences the connection's strength and form. The network's architecture, including the arrangement of nodes, intermediate nodes, their weights w, the threshold w_0, and the nonlinear function φ, determines the network's characteristics [51, 48].

2.3.2 Feedforward networks

Feedforward networks, also known as multilayer perceptrons (MLP), have connections that proceed in a single direction, forming a directed acyclic graph. In this structure, each node receives inputs from preceding nodes and sends information to subsequent nodes without forming cycles [51, 48]. Initially, the weights of each input are assigned randomly and are then adjusted through an iterative learning process. It is established that three layers are sufficient to represent any arbitrary discrimination function, provided the network contains an adequate number of nodes. These three layers are commonly identified as the

input layer, hidden layer, and output layer. Figure 2.9 illustrates a three-layer neural network.

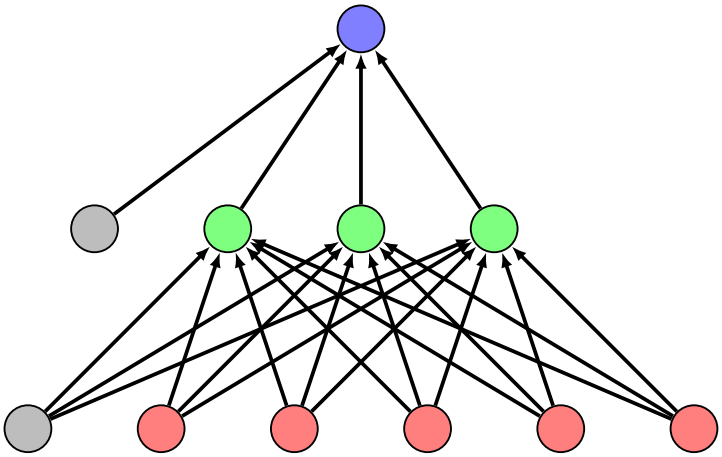


Figure 2.9: Architecture of a multilayer perceptron: input layer (red), hidden layer (green), and output layer (blue).

The learning process in feedforward networks utilizes the backpropagation algorithm. A set of patterns is fed into the network, and information propagates through the layers, with the output y generated by the final layer.

The backpropagation algorithm evaluates the difference between the output y and the desired classification ω. To minimize classification errors, the weights w_{ij} are recalculated and adjusted accordingly [50, 51].

2.3.3 Convolutional neural networks

Convolutional Neural Networks (CNNs) are among the most recognized and widely applied deep learning architectures, inspired by the animal visual system. They have significant applications in computer vision [47]. These networks are characterized by their mesh topology. As a type of feedforward neural network, CNNs transform inputs through a series of sequential layers [48].

The primary distinction between CNNs and feedforward neural networks (FFNNs) lies in the number of layers; CNNs consist of multiple layers compared to the single layer in FFNNs. A typical CNN comprises three types of layers: convolutional layers, pooling layers, and fully connected layers [47, 48].

Figure 2.10 illustrates the general flow of a CNN for image classification tasks. The input is usually a three-dimensional tensor, such as an image with height, width, and

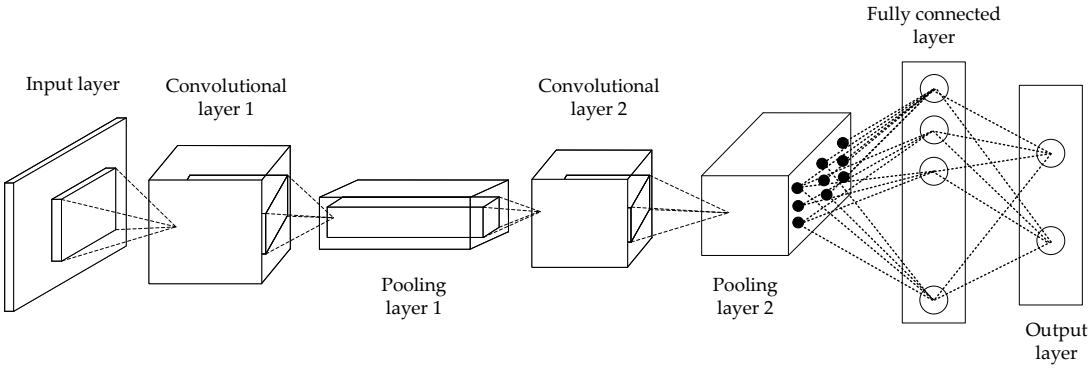


Figure 2.10: Basic architecture of a convolutional neural network.

color channels.

The image is entered at the initial layer and passes through multiple convolutional and pooling layers, which generate representations fed into the final part of the network, the fully connected layer [47].

Convolution layers

Convolutional layers form the core of CNNs, functioning as feature extractors. Each neuron in these layers connects to small regions of the preceding layers, referred to as *receptive fields*, also known as kernels or filters [47]. This operation is typically defined in its discrete form as shown in Equation 2.3.2 Where, I is a two-dimensional image and K is a two-dimensional kernel [48] .

$$S(i,j) = (I * K)(i,j) == \sum_{m}\sum_{n} I(i-m, j-n)\, K(m,n) \tag{2.3.2}$$

Designing convolutional layers involves various parameters, including the number of filters learned from the previous layer. The output of a layer is determined by the filter size, stride size, and padding. The spatial dimension of the filter is known as the kernel size. The stride indicates the step size with which the filter slides over the input to generate the feature map, while padding adds zeros at the edges of the input. After obtaining the feature map, the rectified linear unit (ReLU) activation function is usually applied, which can enhance the CNN's performance [47].

Pooling layers

Pooling layers are employed to achieve spatial invariance translations. Typically, these layers are placed between consecutive convolutional layers to diminish the spatial dimensions of feature maps. Pooling layers serve to progressively reduce the number of parameters and computational load, while also aiding in the mitigation of overfitting. The pooling function operates along the spatial dimension of the input volume, reducing its size while maintaining the depth constant [47, 48].

The most prevalent pooling methods in CNNs are max-pooling and average-pooling. Similar to convolution operations, pooling is conducted with a specified filter size and stride. For instance, the max-pooling operator selects the maximum value within a defined region, whereas the average-pooling operator computes the average of the values in that region

Fully connected layers

Following several cycles of learning through convolution and pooling layers, CNNs extract high-level features from the input data and conduct advanced reasoning based on these representations. Ultimately, fully connected layers are implemented at the network's conclusion. Similar to FFNNs, fully connected layers are one-dimensional, with each neuron in the fully connected layer being connected to every neuron in the preceding layer.

Fully connected layers encompass the majority of the CNN's parameters and impose a substantial computational load during training. Besides the three primary layers (convolutional, pooling, and fully connected), various standardization layers have been incorporated into CNNs, with batch normalization being the most prevalent. Research indicates that batch normalization can accelerate training and reduce sensitivity to weight initialization [47].

2.3.4 Supervised classification

Supervised classification is one of the core tasks in machine learning, where the objective is to assign input data points to predefined categories or classes. The process involves learning a mapping function from input data to corresponding output labels based on a labeled dataset. Each data point in the dataset consists of input features and a target label that indicates the correct class. The goal of supervised classification is to create a

model that can accurately predict the label of unseen data points by generalizing from the patterns learned during training.

The supervised classification process can be divide in two main stages: training and testing. During the training phase, the model is provided with a labeled dataset, where it uses the input-output pairs to adjust its internal parameters, typically via an optimization algorithm such as Stochastic gradient descent (SGD), to minimize a predefined loss function. The loss function quantifies the error between the model's predictions and the true labels. Common loss function for classification is the cross-entropy (see equation 2.3.3). In the testing phase, the trained model is evaluated on a separate set of data (the test set) to assess its generalization ability, which is critical for the model's performance on real-world applications [48].

$$H\left(y, p\right) = -\sum_{i} y_i \log_e\left(p_i\right) \tag{2.3.3}$$

2.3.5 Self-Supervised learning

Self-supervised learning (SSL) has emerged as a powerful paradigm within machine learning, bridging the gap between supervised and unsupervised learning. Unlike traditional supervised learning, which relies on labeled data for training, self-supervised learning leverages unlabeled data by creating supervised data itself. This approach is particularly useful in scenarios where labeled data is scarce or expensive to obtain [52, 53].

SSL revolves around generating pseudo-labels or auxiliary tasks that provide supervisory signals from the data [53]. The core idea is to design pretext tasks that enable the model to learn meaningful representations of the data, which can then be fine-tuned on specific downstream tasks such as classification. Common pretext tasks include predicting missing parts of an image [54], solving jigsaw puzzles [55], or predicting image rotations [56].

In the context of classification, SSL can be particularly advantageous when labeled data is limited . By pre-training models on large amounts of unlabeled data with self-supervised methods, models can learn rich feature representations that improve performance on supervised classification tasks. This transfer of learned representations allows models to achieve higher accuracy with fewer labeled samples [52].

For instance, self-supervised pre-training has been successfully applied to various domains, including computer vision and natural language processing. In computer vision,

techniques like contrastive learning and pretext tasks have demonstrated significant improvements in image classification performance, even with limited labeled datasets [53].

2.4 Discussion

In this chapter, the necessary theoretical bases for the development of this thesis were presented. The topic of neural architecture search and their three most important parts were addressed, then we reviewed the evolutionary techniques in this case with emphasis on Cartesian genetic programming, as well as multi-objective optimization and evolutionary multi-objective algorithms, ending with an introduction to deep neural networks with a focus on convolutional neural networks.

In the following chapter, state of the art for single and multi-objective neural architecture search is analyzed in detail.

Chapter 3

State of the art

In this chapter, we will provide a comprehensive review of the various approaches proposed for neural architecture search. We will explain these approaches from the perspective of evolutionary computation, beginning with a discussion of single-objective methods. These methods focus on optimizing a single metric, typically aiming to improve performance or efficiency. Subsequently, we will explore multi-objective methods that consider multiple criteria simultaneously, such as accuracy, computational cost, or learnable parameters. Through this review, we aim to elucidate the strengths and limitations of each approach, providing a clear understanding of the current state of research in NAS.

Deep architectures, particularly convolutional neural networks (CNNs), have traditionally been designed by experts through manual effort. With the recent exponential growth in computational power, fields like NAS have emerged to leverage these resources, creating methods to improve CNN design. NAS methods can be categorized into two main approaches: evolutionary algorithms (EAs) and reinforcement learning (RL) [9].

The quest to identify high-performance architectures using EAs is not a new challenge; initial efforts date back to the 1990s, with a significant milestone being the proposal of Neuroevolution of Augmenting Topologies (NEAT) in 2002 [23]. NEAT initially worked with small artificial neural networks (ANNs), optimizing both connections and weights. NEAT inspired several approaches, such as HyperNeat [57] and ES-HyperNeat [58].

More recently, the focus has shifted towards Deep Learning, necessitating adaptations in the evolution of these more complex architectures [6, 10].

The fundamental goal of CNNs is to create a feature extractor that, through a series of operations, generates a feature map at each network layer to generalize patterns from the input image, culminating in a classifier. Hence, the connections and operations within base blocks and the overall architecture significantly impact performance.

3.1 Single-Objective NAS

Evolutionary NAS methods focus on optimizing a single metric, most commonly accuracy, to find the best-performing neural networks for a given task. These approaches aim to design models that can maximize performance on a particular dataset, often evaluated through a testing error rate, such as those in CIFAR-10 and ImageNet benchmarks. Several methods have been proposed, achieving competitive results in the design of convolutional neural networks [6, 16, 14]. These methods vary in the encoding schemes they use to represent network architectures and the search strategies that guide the optimization process.

In early approaches, Xie and Yuille [16] introduced Genetic CNN, utilizing a binary encoding scheme for CNN architectures. Here, each bit represented a connection between nodes, corresponding to convolutional operations. Their evolutionary operations included low-probability bit flips and crossover operations, which allowed them to discover patterns similar to established models such as VGG and ResNet. This approach produced a model (GeNet) that achieved competitive accuracy on CIFAR-10, demonstrating that genetic algorithms could effectively navigate complex search spaces for a single objective.

Building on this, CoDeepNEAT [6], the search space was structured by extending the NEAT algorithm to support CNNs, employing coevolutionary strategies to independently evolve two subpopulations of network modules. The objective was to minimize testing error, leading to a competitive 7.3% error rate on the CIFAR-10 dataset. This marked an important development in evolutionary NAS, showcasing the capability of genetic algorithms to find high-performing architectures by minimizing a single objective.

More recently, Suganuma et al. [14] proposed using Cartesian genetic programming for CNN design, with function blocks such as convolutional layers and pooling blocks represented in a graph structure. Their evolutionary strategy applied mutations and evaluated architectures on CIFAR-10 and CIFAR-100 datasets. This approach further advanced single-objective NAS by achieving a 5.01% error rate on CIFAR-10, positioning CGP as a viable method for optimizing CNN architectures with a focus on accuracy, Other CGP based proposal is made by Tobari [59] they introduced a NAS algorithm similar to CGP-CNN. This algorithm employs evolutionary strategies for searching and is block-based. Its main innovation is incorporating a CGP crossover method inspired by the Multiple Sequence Alignment (MSA) algorithm, which aligns amino acid sequences, leading to improvements over traditional CGP.

Additionally, research by Sun et al. [60] introduced variable-length genotype representations using ResBlocks and DenseBlocks, demonstrating that a genetic algorithm with binary tournament selection could reduce error rates while also minimizing model parameters. This technique achieved significant results on CIFAR-10 and CIFAR-100, further proving the effectiveness of single-objective NAS when focused on accuracy optimization.

Despite their success, single-objective NAS methods are limited by their focus, which often overlooks other important factors such as computational efficiency and Parameters. Nevertheless, these methods have proven that evolutionary algorithms can be powerful tools for optimizing CNN architectures under a single objective, often achieving performance comparable to manually designed models [15, 61].

In conclusion, single-objective NAS approaches have demonstrated their ability to optimize neural architectures effectively, achieving competitive results in terms of accuracy across various datasets. However, their focus on a single objective leaves room for further exploration, particularly in balancing other performance metrics such as computational complexity. On the other hand, it can observed that in general the search spaces used were limited to integer or binary representations, where the main focus was trying to represent or approximate already known architectures.

3.2 Multi-Objective NAS

Multi-objective NAS methods aim to optimize several objectives simultaneously, such as accuracy, computational cost or/ and model complexity. One of the earliest examples is the Neuro-Evolution with Multiobjective Optimization (NEMO) approach proposed by [62], which optimizes convolutional neural networks (CNNs) by considering two objectives: error rate and inference time. The network architecture is encoded as an integer vector inspired by the LeNet architecture, where different channel sizes are searched for each layer. The evolutionary process was conducted using NSGA-II, targeting datasets such as MNIST and CIFAR-10, as well as a real-world problem for driver monitoring (drowsy behavior recognition). The authors concluded that NEMO found faster networks without compromising accuracy, although the approach demanded significant GPU power.

In another influential work, [9] proposed NSGANet, a NAS algorithm based on the well-known NSGA-II multiobjective optimization algorithm. NSGANet optimizes two objectives: accuracy and FLOPs, using a binary string encoding similar to [16]. The evolutionary process involves crossover and mutation operators, where offspring are generated

by flipping bits in positions where parent networks differ. NSGANet also incorporates the Bayesian Optimization Algorithm (BOA) [63], which finds relationships among network blocks and routes to promote more efficient exploration of the search space. On the CIFAR-10 dataset, NSGANet achieved a 3.85% error with only 3.34 million parameters and 1290 MFLOPs, while requiring only 8 GPU-days, making it competitive with other state-of-the-art NAS methods.

In [17] a particle swarm optimization (PSO) algorithm for NAS called MOCNN was introduced, which employs a novel representation based on computer networks and IP coding. This approach divides CNN layers into subnetworks, allowing for the inclusion of hyperparameters within the encoding. The authors proposed both single-objective (IPPSO) and multiobjective (MOCNN) versions. The multiobjective approach aimed to optimize accuracy and FLOPs, using DenseNet-121 as a comparison on the CIFAR-10 dataset. MOCNN achieved comparable accuracy, with the authors reporting promising results, although the exact number of FLOPs was not disclosed.

In [64], an evolutionary multiobjective NAS algorithm called MOGIG-Net was introduced, inspired by the encoding of NSGANet [9]. This method also employed weight inheritance, which allows offspring to inherit parent weights, reducing the training time required. The algorithm was evaluated on CIFAR-10 and CIFAR-100 datasets, achieving a classification error of 2.01% with 3.7 million parameters on CIFAR-10, and 14.38% error with the same parameter count on CIFAR-100. Despite the impressive results, the authors did not report the technical specifications used, making it difficult to assess the total GPU-days.

Building upon NSGANet, [10] proposed NSGANetV1, which further optimized classification accuracy and architecture complexity (measured by FLOPs). The evolutionary process was divided into exploration (selection, crossover, and mutation) and exploitation (using a Bayesian network to guide offspring creation). The architecture representation is a directed acyclic graph divided into five main blocks, each with internal nodes performing convolution and pooling operations. On CIFAR-10, NSGANetV1-A4 achieved 97.98% accuracy with 4.0 million parameters, while on CIFAR-100 it obtained 85.62% accuracy with 4.1 million parameters. NSGANetV1 models were also transferred to ImageNet, where the A3 model reached 76.2% Top-1 accuracy.

A further extension, NSGANetV2 [7], introduced surrogate models to speed up both the search and evaluation process. The approach employed an online learning algorithm to estimate fitness values during the search, while a Supernet model enabled weight inher-

itance to reduce training time. NSGANetV2 outperformed other NAS methods on CIFAR-10, achieving 98.4% Top-1 accuracy, while also producing less complex models. For ImageNet, the method reached 75.9% Top-1 accuracy, demonstrating its ability to balance multiple objectives efficiently.

Other notable contributions include MOGI-NET [65], which also employed weight inheritance to reduce training epochs, and EEEA-NET [12], which introduced a novel population initialization mechanism that considers a parameter threshold to reduce total search time. LF-MOGP [66] used Cartesian genetic programming with a leader-follower evolution mechanism to maintain an external archive of the best solutions, promoting diversity in the search. EvoApproxNAS [67] extended CGP by incorporating bottleneck and inverted residual blocks to simultaneously optimize power consumption, network parameters, and classification error.

3.3 Discussion

Based on the state-of-the-art review, we found that the main challenges in this area are the following: More search spaces need to be explored, as well as the involvement of different representations, such as the use of real or continuous domains. The inclusion of hyperparameters is also barely used in the revised search spaces. We can observe that the use of Cartesian genetic programming has gained significant attraction, but the representation has not been extended or modified, as well as that it is just beginning to be tested in multi-objective approaches. On the other hand, we can observe that both search and time estimation strategies are slightly addressed or already established methods are used. But, it is necessary to approach new methods, which can better integrate with evolutionary computation, as well as generate a synergy with these methods.

Therefore, our thesis work fill the gaps generated from the state of the art by proposing a multi-objective approach using Cartesian genetic programming, as well as using a continuous search space, which can be adaptable to different MOEAs. Finally, we also propose new strategies to perform the search, as well as to estimate the performance, in order to address the current problems of NAS. Finally in Table 3.1 it is shown a summary of the state-of-the art related algorithmic proposals.

Table 3.1: Reviewed works of the state of the art

Year	Name	Application	Benchmark	EC	Encoding	Model	Objetive	Proposal
2017	A genetic programming approach to design convolutional neural network architectures [68]	Image Classification	CIFAR-10	CGP, ES	Graph, Integer	CNN	Accuracy	CGP-CNN
2017	NEMO : Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy [62]	Drowsiness Recognition for Driver Monitoring System	CIFAR-10, MNIST	NSGA-II	Integer	CNN	Accuracy Inference Speed	NEMO
2017	Genetic CNN [16]	Image Classification	CIFAR-10 ILSVRC2012	GA	Binary	CNN	Accuracy	GeNet
2017	NEMO : Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy [62]	Drowsiness Recognition for Driver Monitoring System	CIFAR-10, MNIST	NSGA-II	Integer	CNN	Accuracy Inference Speed	NEMO
2019	Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming [14]	Image Classification	CIFAR-10, CIFAR-100	CGP, ES	Graph, Integer	CNN	Accuracy	CGP-CNN
2019	NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm [9]	Image Classification	CIFAR-10	NSGA-II	Binary	CNN	Accuracy FLOPs	NSGA-Net
2019	A Graph-Based Encoding for Evolutionary Convolutional Neural Network Architecture Design [69]	Image Classification	CIFAR-10	Random search	Graph	CNN	None	Random search
2019	Evolving Deep Neural Networks [6]	Image Captioning	CIFAR-10	GA	NEAT-DNN	CNN, LSTM	Accuracy	CooDeepNeat DeepNeat
2020	NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search [10]	Image Classification	Aircraft, CIFAR-10, CIFAR-100, CINIC-10 DTD, Flowers102, ImageNet, Pets, STL-10	NSGA-II	Integer	CNN	Accuracy MAdds	MSuNAS NSGA-NetV2

Table 3.1: Reviewed works of the state of the art

Year	Name	Application	Benchmark	EC	Encoding	Model	Objetive	Proposal
2020	Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification [7]	Image Classification	CIFAR-10, CIFAR-100 ImageNet	NSGA-II	Block	CNN	Accuracy FLOPs	NSGA-NetV1
2020	Designing Convolutional Neural Network Architectures Using Cartesian Genetic Programming [14]	Image Classification Inpainting Denoising	CIFAR-10, CIFAR-100	CGP, ES	Graph, Integer	CNN	Accuracy	CGP-CNN
2020	Particle Swarm Optimization for Evolving Deep Convolutional Neural Networks for Image Classification: Single and Multi-Objective Approaches [17]	Image Classification	CIFAR-10, Convex Sets MNIST Basic, MRDBI	OMOPOSO PSO	IP-based	CNN	Accuracy FLOPs	IPPSO MOCNN
2020	Fast Evolution of CNN Architecture for Image Classification[15]	Image Classification	CIFAR-10, CIFAR-100 SVHN	GA	Integer	CNN	Accuracy	GAnet
2020	Completely Automated CNN Architecture Design Based on Blocks [60]	Image Classification	CIFAR-10, CIFAR-100	GA	Block	CNN	Accuracy	AE-CNN
2021	A Multi-Objective Evolutionary Approach Based on Graph-in-Graph for Neural Architecture Search of Convolutional Neural Networks [65]	Image Classification	CIFAR-10, CIFAR-100	NSGA-II	Block	CNN	Accuracy Parameters	MOGIG-Net
2021	EEEA-Net: An Early Exit Evolutionary Neural Architecture Search [12]	Image Classification	CIFAR-10, CIFAR-100 ImageNet	NSGA-II	Integer	CNN	Accuracy Parameters	EEEA-Net
2022	Using Cartesian Genetic Programming Approach with New Crossover Technique to Design Convolutional Neural Networks [59]	Image Classification	CIFAR-10, CIFAR-100	CGP, ES	Graph, Integer	CNN	Accuracy Parameters	MSA

Table 3.1: Reviewed works of the state of the art

Year	Name	Application	Benchmark	EC	Encoding	Model	Objetive	Proposal
2022	Evolutionary convolutional neural network for image classification based on multi-objective genetic programming with leader–follower mechanism [66]	Image Classification	CIFAR-10, Fashion, CIFAR-100, MNIST	NSGA-II	Graph, Integer	CNN	Accuracy Parameters	LF-MOGP
2022	Evolutionary approximation and neural architecture search [67]	Image Classification	CIFAR-10, SVHN	NSGA-II	Graph, Integer	CNN	Accuracy Energy	EvoApproxNAS

Chapter 4

Continuous Representation for Multi-objective NAS

In this chapter, we introduce a two-level solution representation encoding for CNN architecture search. The proposals are CGP-NASV1 and CGP-NASV2. At the first level, we employ the CGP representation using acyclic graphs that considered feed-forward connections, which are represented as integer-based vectors. At the second level, these CGP-based solutions are converted into real-valued vectors.

The motivation behind this proposed representation is to improve the multi-objective neural architecture search by utilizing a continuous representation based on Cartesian Genetic Programming. The goal is to demonstrate the benefits of continuous representations compared to discrete ones, particularly for optimizing neural architectures in multi-objective contexts. Additionally, in this chapter we examine the performance of various multi-objective evolutionary algorithms to better understand how different algorithms impact the quality and diversity of the resulting solutions.

Moreover, in this chapter we investigate a block-chained representation combined with the proposed continuous approach to simplify the complexity of the search space. By integrating these representations, the aim is to enhance the flexibility and adaptability of the search process.

4.1 Multi-objective NAS problem

In this chapter, we propose a representation for the design of CNN architectures that simultaneously maximize accuracy and minimize complexity. Our objective is to constrain

model capacity to achieve accurate models with moderate complexity. Our method utilizes a CGP representation, enabling operations in the real domain and facilitating the use of standard multi-objective optimization techniques. The problem can be formulated as a multi-objective optimization task as follows:

$$\text{Minimize } \mathbf{F}(x) = \left(f_1\left(x; w^*(x)\right), f_2(x)\right)^T \tag{4.1.1}$$

$$\text{subject to: } w^*(x) \in \operatorname{argmin} \mathcal{L}(w; x) \tag{4.1.2}$$

Here, f_1 represents the classification error of the CNN architecture defined by parameters w^*, and f_2 represents model complexity, measured in MAdds (the total number of operations performed in each architecture). Previous studies have shown that MAdds are a useful guideline for specific implementation scenarios, such as mobile settings, where complexity should not exceed 600 MAdds [10, 70]. To estimate classification error, it is necessary to optimize the weights w of the CNN architecture, making x (solution) dependent on this optimization, typically performed using algorithms like stochastic gradient descent (SGD).

4.2 Solutions representation for CNN - Search space

Our initial idea is to use CGP, as explained in Section 2.2.2. CGP relies on an integer-based vector to represent the genotype. To modify this representation at the genotype level, we utilized the approach proposed by Clegg et al [71].

Equation 4.2.1 defines a range where $func_k$ is the current function identifier and $func_{total}$ is the total number of functions in the function set. Within this range, a uniform random number is generated to represent a function in the real domain.

$$rfunc_k \in \left[\frac{func_k}{func_{total}}, \frac{func_k + 1}{func_{total}}\right] \tag{4.2.1}$$

Equation 4.2.2 defines the range for the input connections in the real domain to map each node connection. An input value ($nodeinput$) and its node number ($nodeTerm$) are used to calculate this range.

$$rinput_j \in \left[\frac{nodeinput_j}{nodeTerm}, \frac{nodeinput_j + 1}{nodeTerm} \right] \tag{4.2.2}$$

Equation 4.2.3 decodes the function identifier by multiplying the gene value by the total number of functions, and Equation 4.2.4 obtains the value of each connection by multiplying the gene value and the node number. It should be noted that the real-based or integer-based vector must be divided into segments. The leftmost position per segment represents the function, and the other values represent the connections.

$$func_k = \lfloor gene_i \times func_{total} \rfloor \tag{4.2.3}$$

$$input_j = \lfloor gene_i \times nodeTerm \rfloor \tag{4.2.4}$$

4.2.1 CGP Funtion set

CGP operates on a set of functions that implicitly define the search space, as shown in Table 4.1. For CGP-NASV1 and CGP-NASV2, we considered 11 functions, making explicit all possible combinations of the number of channels and kernels.

The ConvBlock [68] function is a basic block composed of three stages: convolution, batch normalization, and the ReLU function. This function is illustrated in Figure 4.1a.

On the other hand, the ResBlock [72] function involves another significant component of convolutional networks, the shortcut connections. As shown in Figure 4.1b, it internally contains a ConvBlock, an additional convolution operation, and a summation operation that performs a tensor addition of both inputs, the original input and the preprocessed input. It is important to note that a 1×1 convolution operation is performed to match the feature map sizes of both inputs before the tensor addition.

The Bottleneck block [73] (Figure 4.1c) imply a reduction in the number of parameters and computations. By using the first 1×1 convolution to reduce the dimensionality, the computationally intensive $k \times k$ convolution operates on fewer channels, thus decreasing the overall computational cost. Another 1×1 convolution is used to restore the original number of channels.

The Fused-MBConv [73] (Figure 4.1d) block with Squeeze-and-Excitation [74] combines efficient convolution operations and adaptive feature recalibration. It starts with a

Table 4.1: Functions set with corresponding variations and arity

Block type	Symbol	CGP-NASV1	CGP-NASV2	Arity
ConvBlock	$CB(C,k)$	$C \in \{32,64,128,256\}$ $k \in \{1\times1, 3\times3, 5\times5, 7\times7\}$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	1
ResBlock	$RB(C,k)$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5, 7\times7\}$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	1
Bottleneck	$BN(C,k)$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5, 7\times7\}$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	1
FusedMBconv	$FBC(C,k)$	$C \in \{32,64,128,256\}$ $k \in \{3\times3\}$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	1
MBconv	$MBC(C,k)$	Not used in CGP-NASV1	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	1
SepConv	$SC(C,k)$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5, 7\times7\}$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	1
DiConv	$DC(C,k)$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	$C \in \{32,64,128,256\}$ $k \in \{3\times3, 5\times5\}$	1
Indentity	I	-	-	1
C1x7-7x1	$C17$	-	-	1
Summation	Sum	-	-	2
Concatenation	Concat	-	-	2

fused convolution that expands the channels followed by batch normalization and activation. The SE block then scales the feature map channels adaptively using global average pooling and a set of fully connected layers. Finally, a 1x1 convolution restores the original number of channels. This architecture enhances efficiency and representational power.

The MBConv block [75] (Figure 4.2c), essential in MobileNetV2, optimizes computational efficiency and performance for mobile and embedded devices. It comprises three main convolutional steps: an initial 1x1 pointwise convolution to reduce dimensionality, a depthwise convolution to handle spatial relationships within channels, and a final 1x1 pointwise convolution to restore dimensionality. The Squeeze-and-Excitation module is included to recalibrate feature maps. A skip connection is used to improve gradient flow and mitigate the vanishing gradient problem, ensuring efficient training and high accuracy with fewer parameters.

The SepConv function applies the Separable Convolution [76] block (Figure 4.2a), this enhances efficiency in CNNs by separating spatial and channel-wise convolutions. It consists of a Depthwise Convolution, which applies a single filter per input channel, followed by a Pointwise Convolution (1x1), which combines these outputs. This architecture reduces the number of parameters and computational cost compared to standard convo-

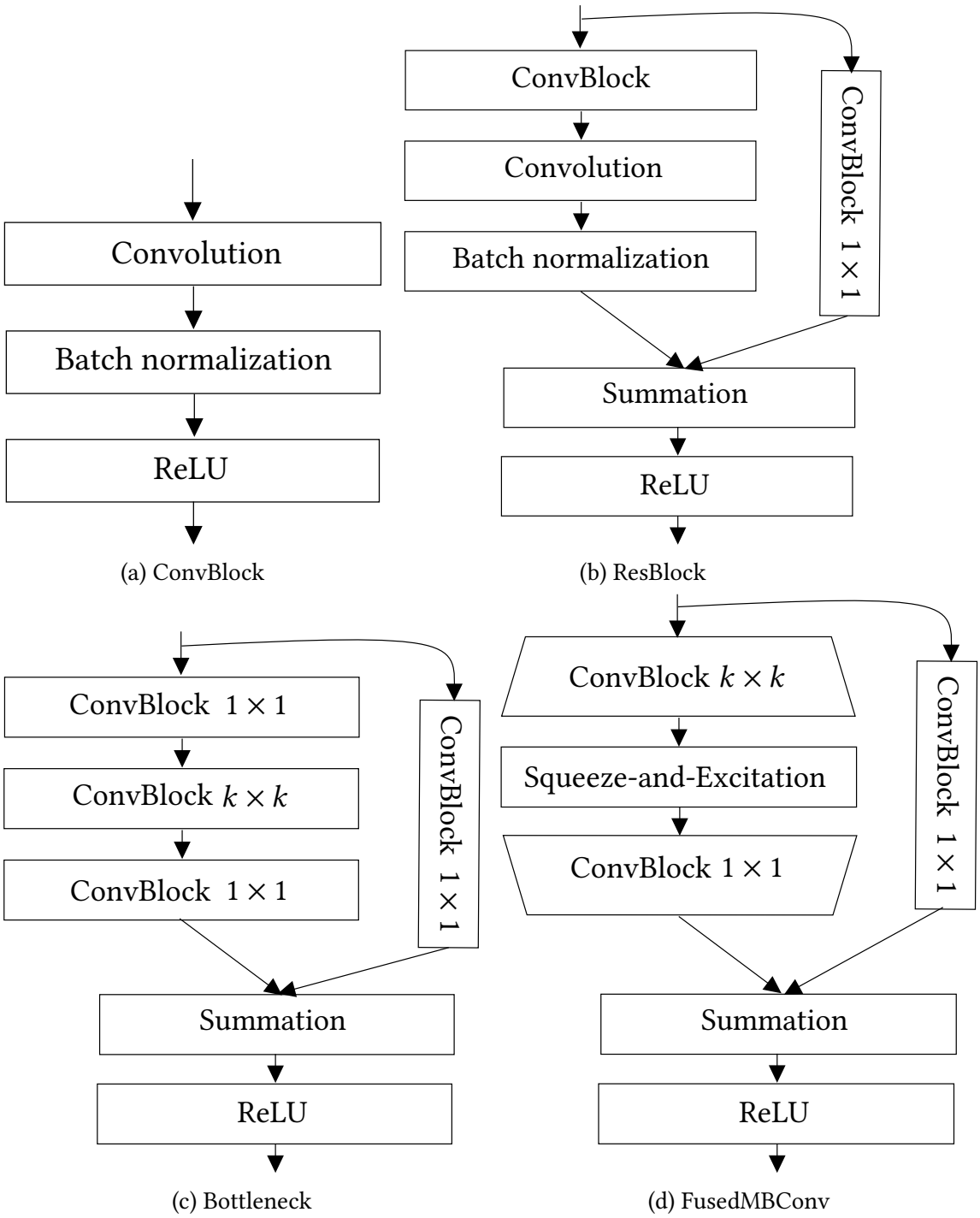
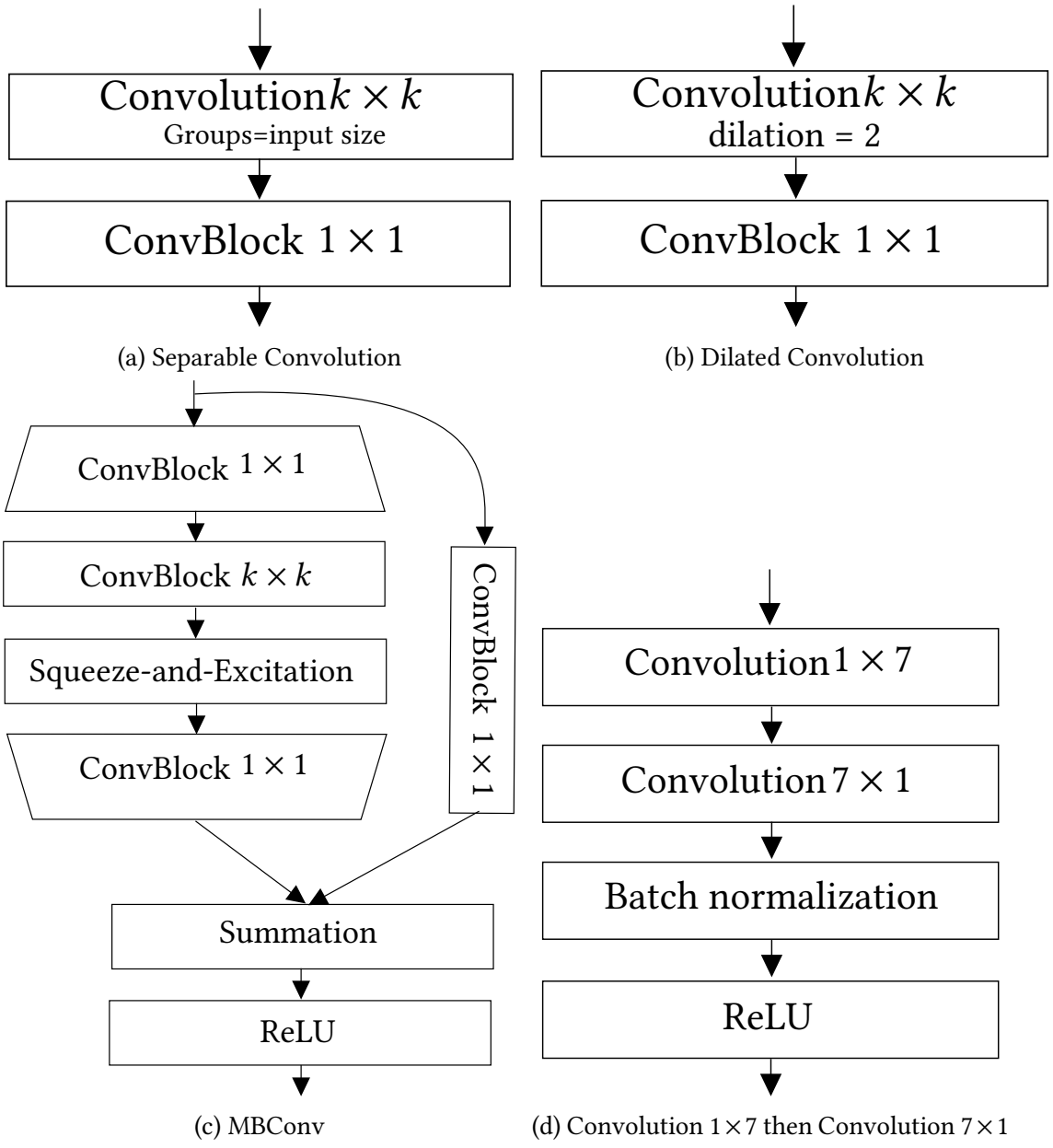


Figure 4.1: Function set blocks.



(a) Separable Convolution (b) Dilated Convolution

(c) MBConv (d) Convolution 1×7 then Convolution 7×1

Figure 4.2: Function set blocks Cont.

lutions, maintaining performance while being more efficient. Separable convolutions are used in efficient CNN architectures such as MobileNet [76] and Xception [77].

The Dilated Convolution [78] or DiConv block (Figure 4.2b), increases the kernel's receptive field by inserting "holes" between elements in convolution, allowing for better

context capture without changing kernel size or parameters. They retain image resolution while eliminating the loss caused by pooling. In our implementation, we also used a 1×1 convolution after dilated convolution because it can reduce the number of channels, which helps in reducing the computational load and the number of parameters in the model.

The C$1 \times 7 - 7 \times 1$ block [70] is a sequence of applying a 1×7 convolution followed by a 7×1 convolution (Figure 4.2d), is an efficient technique to approximate a larger 7×7 convolution, significantly reducing computational cost and parameter count while preserving spatial information. This approach captures horizontal and vertical features separately, enhancing feature extraction in CNN architectures. Compared to a direct 7×7 convolution, this method uses fewer parameters and computations, making it suitable for resource-constrained environments.

The identity, summation, and concatenation blocks are simple functions that facilitate connections between other blocks, thereby increasing the diversity of possible architectures. The identity block [70] acts as a bridge between two blocks, helping to reduce the overall number of operations. On the other hand, the Summation and Concatenation blocks [14] are the only ones with arity 2, meaning they receive two tensors. In the case of summation, an element-wise summation of two feature maps is performed channel by channel, resulting in a new tensor of the same size. Finally, the concatenation block concatenates two inputs along the channel dimension, producing a new feature map with a size equal to the sum of the sizes of the two input feature maps.

4.2.2 CGP-NASV1 representation

Using the aforementioned as a basis, we can define a Cell-based representation as explained in Section 2.1.1, With this information, we can introduce CGP-NASV1, which implements a block-chained approach. see Figure 4.3. In each "Normal" block, an internal CGP representation itself handles the connections and functions, in the "Reduction" blocks, a spatial reduction is applied, in our case, a pooling block is fixed. During the optimization process, this representation is re-encoded to a real-valued representation. Therefore, the full MOEAs searching potential within continuous domain is exploited. CGP-NASV1 employs block-chained encoding at the highest level, defining a template with multiple layers that connect blocks sequentially, each fulfilling specific tasks. Spatial reduction in CGP-NASV1 is achieved through pooling layers. At the top level of the block-

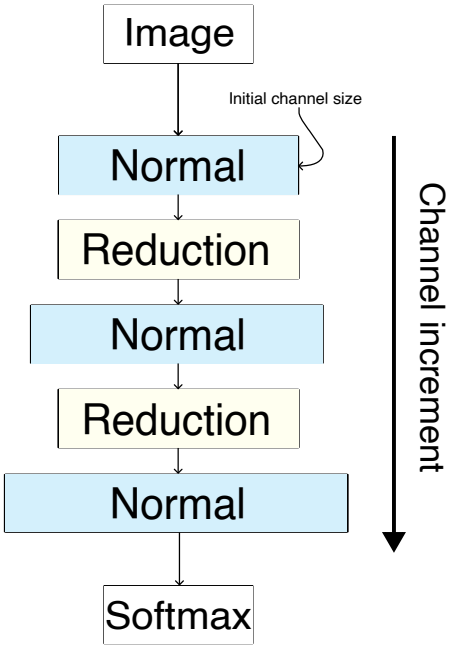


Figure 4.3: The general scheme of the representation based on chained blocks; each block internally focuses on special operations; in each normal block, for example, convolution operations are performed; on the other hand, in the reduction blocks, methods such as pooling are applied to spatially reduce the feature maps. As the number of blocks increases, the number of channels will gradually increase.

chained structure, CGP-NASV1 integrates a CGP within a "Normal" block, while reduction blocks incorporate max pooling layers. All pooling blocks maintain a fixed size with a 2×2 kernel and a stride of 2. Figure 4.4 illustrates an example where, following the final block, global average pooling and a fully connected layer are added. Similar methodologies are endorsed in related literature reviews [7, 67]. Figure 4.4 depicts the representation as an integer-based vector. Each square encapsulating a function symbolizes a "Normal" block with its own CGP configuration, including distinct function sets and sizes. Function IDs are highlighted in red within the figure. CGP-NASV1 represents CNN architectures in a divide and conquer approach. The use of a block-chained schema in synergy with CGP allows more control over the final architecture.

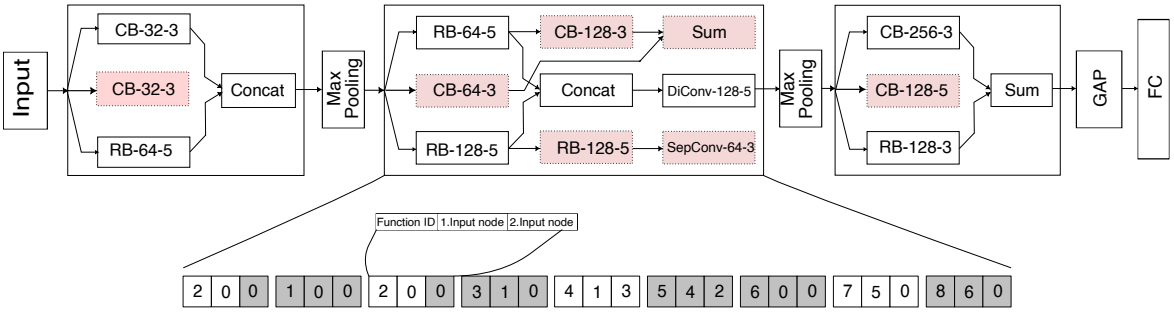


Figure 4.4: CGP-NASV1 using the block-chained representation.

4.2.3 CGP-NASV2 representation

The CGP-NASV2 proposal incorporates hyperparameters into the solution representation used in CGP-NASV1, allowing them to be evolved and optimized during the search process. Modifications in CGP-NASV2 are applied to the block-chained "low level" encoding, leading to a reduction in the number of functions within the function set. However, the inclusion of hyperparameters in the search process inevitably increases the size of the integer vectors (and the corresponding real-based vectors) used to encode the block-chained solutions.

In deep neural architectures, hyperparameters such as the number of channels or filters and kernel size play a crucial role. In CGP, these hyperparameters are explicitly included in the function set if not already considered within the solution representation and evolutionary search. Therefore, a comprehensive function set must be defined to encompass all possible combinations of these hyperparameters.

Moreover, a larger function set directly increases the CGP grid size required to accommodate them uniformly.In CGP-NASV2, hyperparameters are explicitly added to the vector representing the CGP. This approach reduces the number of functions, as only the standard functions remain without the need for their hyperparameter configurations.

In Figure 4.5, a possible solution in CGP-NASV2 is shown. The number of channels and kernel size are defined as parameters associated to each block-chained. The idea is to associate these parameters as weights to each CGP node. Green positions at the integer vector encoding in Figure 4.5 represent the assigned hyperparameters within their corresponding CGP block.

Figure 4.6 illustrates an example of a CGP-NASV2 solution representation as an inte-

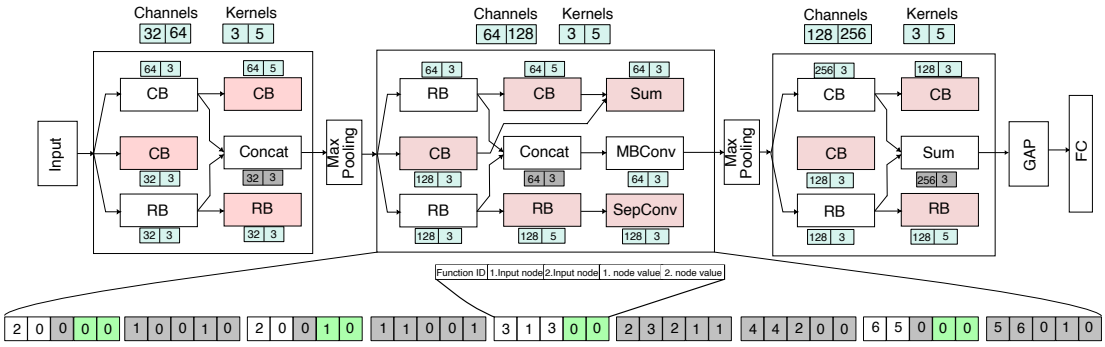


Figure 4.5: CGP-NASV2 block-chained representation with the hyperparameters directly encoded.

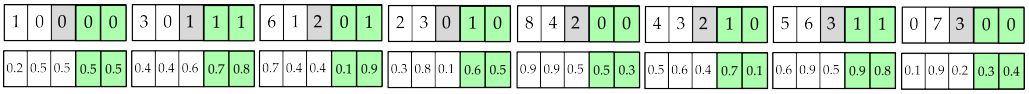


Figure 4.6: A CGP-NASV2 solution represented as an integer vector and its equivalent real-based vector with the added hyperparameters in the green positions.

ger vector with hyperparameters. The entire vector is altered when hyperparameters are explicitly considered by the CGP. The first three positions (white and gray) represent the function ID and input connection nodes. The green positions indicate two hyperparameters: one for the number of channels and the other for the kernel size. For instance, in the integer vector representation shown in Figure 4.5, a 0 in position 4 signifies a channel value of 64, and a 1 in position 5 indicates a kernel size of 5x5.

To convert the integer vector to a real-based vector, the same mechanism explained in section 4.2 is applied, with the addition of Equations 4.2.5 and 4.2.6 to encode and decode the hyperparameters.

$$rHyp_k \in \left[\frac{Hyp_k}{Hyp_{total}}, \frac{Hyp_k + 1}{Hyp_{total}} \right] \tag{4.2.5}$$

$$Hyp_j = \lfloor gene_i \times Hyp_{total} \rfloor \tag{4.2.6}$$

The key difference between the two proposed variants is that in CGP-NASV1, the function set is limited to explicit instantiations of the functions. In contrast, CGP-NASV2 does not have this restriction, requiring the algorithm to determine which functions and hy-

perparameters (filter size, channels) to use when building solutions. This approach offers greater flexibility at the cost of expanding the search space.

When we compare this new representation with others in the state of the art, we can highlight the use of this two-level representation, in this case the use of real encoding, letting us adapt any MOEA for NAS; this characteristic is not found in other methods; on the other hand, this representation can be adapted to multiple hyperparameters; and finally, the use of CGP gives us a variable length representation because we can take advantage of the nature of the inactive nodes in CGP, letting us to obtain a diversity of solutions.

4.3 Evolutionary algorithm - Search strategy

Adopting real-valued representations allows us to leverage any MOEA that operates in such a domain as a search strategy. Consequently, we initially selected the NSGA-II MOEA as our optimizer, given its widespread use and effectiveness. Figure 4.7 presents a general schema of this well-known MOEA. Since both CGP variants represent solutions in the continuous domain for the search process, NSGA-II's full capabilities are preserved. The only significant modifications are in the crossover and mutation operations, which were adapted for CGP-NASV1 and CGP-NASV2 due to the block-chained representation.

Each individual, represented as a real-based vector, is splitted into three sub-vectors, each encoding a CGP "Normal" block. Figure 4.8 illustrates the application of crossover and mutation operations. The crossover operation is performed between sub-vectors at the same overall position within the full solution vector, while the mutation operation is applied independently to the offspring of each sub-vector. It is crucial to note that each operation is carried out independently at the sub-vector level. In the example shown, three crossover and mutation operations are performed for each sub-vector's offspring (pairs of CGP "Normal" blocks), which together form the complete neural architecture to be evaluated. The crossover and mutation operators are those commonly used in NSGA-II: simulated binary crossover (SBX) and polynomial mutation (PM).

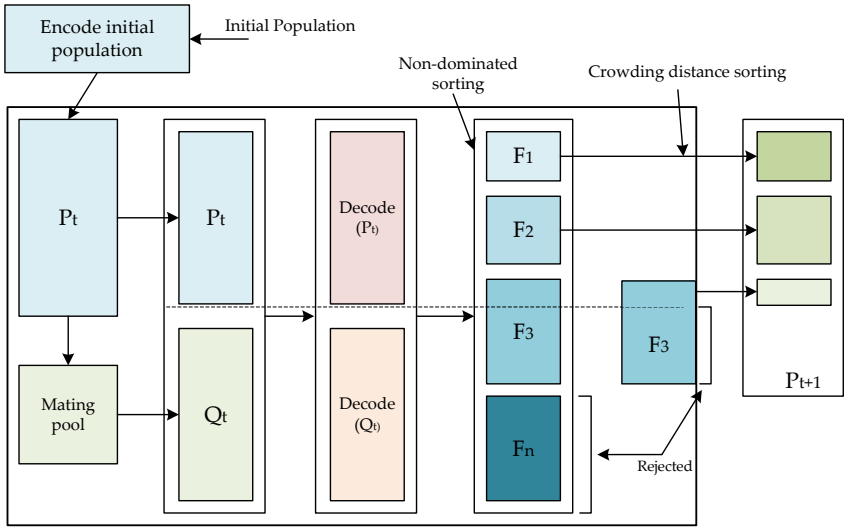


Figure 4.7: NSGA-II general schema with encoding and decoding steps. NSGA-II generates an initial population *Pt*, and after mating, an offspring population *Qt* is obtained. *Qt* is decoded for evaluation and optimization considering the non-dominance criterion.

4.4 Fitness function - Performance estimation strategy

As previously mentioned, in the introduction of this chapter we aim to simultaneously optimize two objectives with both variants of CGP-NAS: accuracy and complexity as measured in MAdds. The goal is to maximize accuracy while minimizing complexity.

The accuracy estimate is obtained by evaluating the validation set, which is a random subset of the training set. This method of obtaining accuracy is referred to as the partial training performance estimation strategy (see Section 2.1.3).

To align both objectives for minimization, the classification error is calculated using the expression 1 − Accuracy. A CNN architecture relies on the convolution operation, which primarily involves multiplications and additions. Therefore, complexity is calculated as the total number of multiplication and addition operations performed by the CNN. The complexity, calculated as the sum of MAdds, is the second objective to minimize, to obtain such data, the PyTorch Profiler was used, which is part of the PyTorch API.

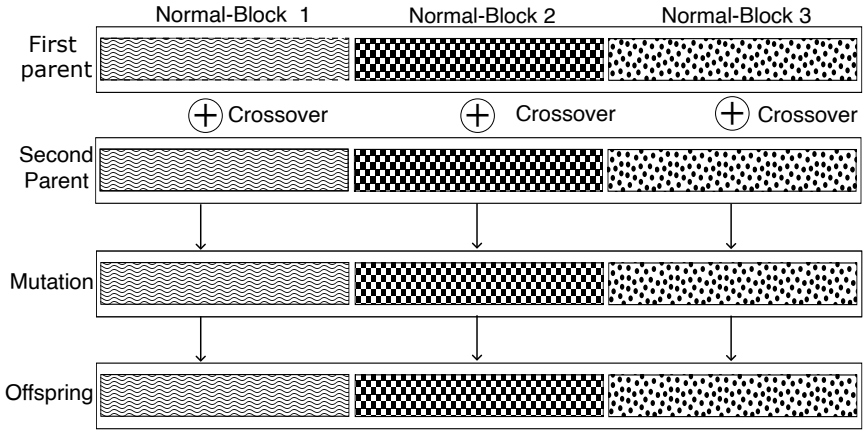


Figure 4.8: Each sub-vector represents a CGP "Normal" block. Crossover is applied independently between sub-vectors at the same overall position, while mutation operates on the sub-vector offspring.

4.5 Experimental framework

To evaluate the proposed CGP-NASV1 and CGP-NASV2 approaches, we first conduct a direct performance comparison between the best evolved architectures using accuracy as the performance metric. Subsequently, a multiple-criteria decision analysis is performed to identify CNN models that best balance both objectives: accuracy and complexity. This section outlines the experimental settings for these evaluations and describes the datasets used.

Table 4.2: CGP-NASV1 and CGP-NASV2 parameters

Parameters	**CGP-NASV1**	**CGP-NASV2**
Rows	5	10
Columns	25	4
Level-Back	1	1
Mutation probability	$Pm = 0.3$	
Crossover probability	$Pc = 0.9$, distribution index for simulated binary crossover $Dsc = 20$.	
Population	24	24
Generations	30	30

Table 4.2 presents the configurations for CGP-NASV1 and CGP-NASV2. The key modification in CGP-NASV2 is the direct representation of hyperparameters within the solu-

tions, requiring a smaller grid size due to a reduced function set. In contrast, CGP-NASV1 needs a larger grid due to the comprehensive function set required to express all possible hyperparameter combinations.

For evaluating the proposed algorithms, we used three image classification datasets: CIFAR-100 [79], CIFAR-10 [79], and SVHN [80]. CIFAR-100 consists of 100 classes, and CIFAR-10 consists of 10 classes, with each dataset containing 50,000 training images and 10,000 test images of 32×32 pixels. Figure 4.9 shows sample images from these datasets. The SVHN dataset includes 99,289 images, with 73,257 for training and 26,032 for testing, also at 32x32 pixels.

During the evolutionary search, solutions were evaluated using a training set split randomly into 80% for training and 20% for validation. For the final evaluation of selected solutions from the Pareto front, the entire training dataset was used to train the models, which were then evaluated on the test partition.

The CNN architecture was trained using Stochastic Gradient Descent (SGD) with a cosine annealing learning rate schedule. The initial learning rate was set to 0.025, momentum to 0.9, and weight decay to 0.0005. The batch size was set to 128, and 36 training epochs were executed during the evolutionary search. Data was preprocessed with a 4 pixel-mean subtraction padding and randomly cropped to 32×32 patches or their horizontally flipped versions. Final solutions were retrained for 600 epochs with cutout preprocessing, and the batch size was set to 96. An auxiliary head classifier [9] was used to enhance training, concatenated after the second reduction block, with its loss multiplied by 0.4 and added to the main architecture's loss.

All experiments were executed on a Supercomputing Node with 2 Intel Xeon E5-2650 v4 @ 2.20GHz processors, 8 Nvidia GTX 1080 Ti GPU cards, and 128GB of RAM, running on Centos 7 OS.

4.6 Results analysis

We designed three experimental setups to evaluate the proposed representation approaches. The difference between the first two experiments lies in the method used to select the final solution from the achieved Pareto front. In the first experiment, the final solution is determined by the smallest classification error, following a criterion similar to

(a) CIFAR-10

(b) CIFAR-100

(c) SVHN

Figure 4.9: Examples from the CIFAR-10, CIFAR-100, and SVHN datasets, with each row representing a class.

that used in most state-of-the-art works. Hence, we selected the evolved CNN presenting the lowest classification error. This solution is then executed 10 times to facilitate a fair comparison with state-of-the-art solutions and other discussed algorithmic approaches using three datasets: CIFAR-10, CIFAR-100, and SVHN.

In the second experiment, CGP-NASV1 and CGP-NASV2 were compared using a multiple-criteria decision-making (MCDM) method to identify the solution with the best Pareto front trade-off on the CIFAR-10 and CIFAR-100 datasets.

In the third experiment, CGP-NASV2 was assessed by implementing other MOEAs and search operations. To verify the developed representation, three different MOEAs were used: NSGA-II with the differential evolution crossover operator, MOEA/D, and SMS-EMOA, using the CIFAR-100 dataset for comparison.

Finally, we compare both proposed approaches against state-of-the-art works using the three datasets considered and discuss how the proposed solution representations and search mechanisms impact the final performances of CGP-NASV1 and CGP-NASV2.

4.6.1 Effectiveness of searching for the hyperparameters

In the first experiment we empirically assessed the effectiveness of including the hyperparameters in the evolutionary search. Thus, we compared CGP-NASV1 versus CGP-NASV2 on the CIFAR-10 and CIFAR-100 datasets. The best evolved architectures in terms of accuracy from 10 independent runs were evaluated. We also added CGP-NAS [81] as a baseline for comparison because it uses the same mechanism for selecting solutions from the Pareto front.

Table 4.3: Comparison between differents versions of CGP-NAS, on CIFAR-10 and CIFAR-100 datasets, parameters and MAdds expressed in millions (1×10^6).

	CIFAR-10			CIFAR-100		
	Error	Parameters	MAdds	Error	Parameters	MAdds
CGP-NAS [11]	4.86 (5.42 ± 0.46)	1.40 (2.52 ± 0.90)	388.71 (1167.13 ± 477.11)	24.23 (26.41± 1.41)	5.43 (5.89 ± 2.75)	1581.93 (1229.11 ± 782.00)
CGP-NASV1	4.23 (4.73 ± 0.44)	8.47 (8.74 ± 3.27)	1255.93 (1161.09 ± 373.78)	21.76 (24.20 ± 1.80)	3.60 (7.25 ± 3.24)	791.85 (792.02 ± 342.60)
CGP-NASV2	**3.70** **(4.07 ± 0.17)**	4.04 (5.82 ± 2.70)	636.32 (818.61 ± 372.62)	**20.63** **(22.49 ± 1.04)**	5.90 (6.50 ± 1.70)	827.00 (850.74 ± 476.08)

Table 4.3 shows the performance effect of evolving the hyperparameters. CGP-NASV2 reduces the error by half a percentile point compared to CGP-NASV1 and by more than

one percentile point compared to the baseline approach CGP-NAS. This reduction is also observed in the standard deviation corresponding to the 10 experimental samples, demonstrating that CGP-NASV2 is robust compared to the other approaches.

We can also observe that the complexity measured in MAdds achieved by CGP-NASV2 is lower than that obtained by CGP-NASV1 for the CIFAR-10 dataset. On the other hand, both have similar values in MAdds for the CIFAR-100 dataset. It is worth mentioning that the number of parameters is not one of the objectives to optimize, hence a significant variation among results is observed. Selecting the best evolved architecture by accuracy shows that CGP-NASV2 achieves the best results in terms of classification error. In contrast, the complexity objective is negatively affected, evidenced by a higher standard deviation.

4.6.2 Best trade-off solution via multiple-criteria decision analysis

CGP-NASV1 and CGP-NASV2 are multi-objective proposals; therefore, selecting the best solution in terms of accuracy from the Pareto front is not an efficient criterion. When evolving neural architectures as a multi-objective optimization problem, the best solution would represent a trade-off between objectives. In the proposed algorithmic approaches, this trade-off is between the classification error and the complexity of the architecture in terms of MAdds. The Multi-Criteria Decision Making (MCDM) method was used to select the best solution from the Pareto front. In particular, the *Knee and Boundary Selection* method [82] has been explored for the empirical analysis.

This method obtains two solutions, called boundary heavy and boundary light, corresponding to the solution with the lowest value per objective. After, it calculates the solution closest to the intersection of both solutions (heavy and light). The obtained solution is called the knee and represents the solution with the best trade-off. Figure 4.10 shows how the knee and boundary selection works.

Table 4.4 shows the results achieved after applying the MCDM analysis to compare CGP-NASV1 and CGP-NASV2 in both CIFAR-10 and CIFAR-100 datasets.

The differences found with this selection method mostly favor solutions with a lower number of parameters and MAdds. Although the classification error increased compared to selecting the solution with the lowest classification error, the difference is relatively minor, averaging a 2% increase on CIFAR-10 and a 5% increase on CIFAR-100. However, the reduction in MAdds from 818.61M to 79.44M for CIFAR-10 and from 850.74M to 55.66M

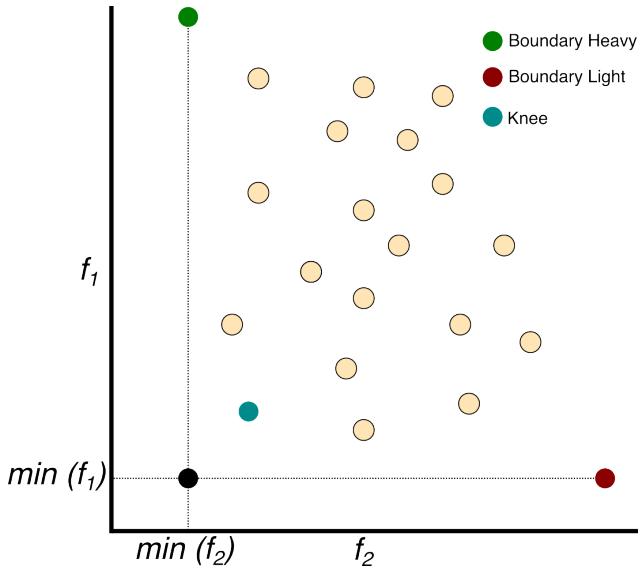


Figure 4.10: Knee and boundary selection method. Measuring the distance between the intersections of the two best extreme solutions ensures the best trade-off solution is obtained.

Table 4.4: Trade-off knee solutions from the Pareto front. Parameters and MAdds are expressed in millions (1×10^6).

	CIFAR-10			CIFAR-100		
	Error	Parameters	MAdds	Error	Parameters	MAdds
CGP-NASV1	5.66 (7.33± 1.55)	0.41 (0.36 ± 0.13)	70.13 (52.01 ± 21.09)	31.47 (33.34 ± 1.60)	1.06 (0.82 ± 0.52)	45.70 (30.81 ± 9.70)
CGP-NASV2	**4.85 (5.59± 0.50)**	0.78 (0.71 ± 0.31)	53.99 (79.44 ± 31.96)	**23.57 (28.23 ± 2.20)**	0.49 (0.53 ± 0.13)	66.66 (55.66 ± 19.14)

for CIFAR-100 represents a substantial decrease in model complexity. This highlights the ability to find solutions that balance both objectives. Additionally, our results indicate that incorporating the hyperparameters within the search process is beneficial, particularly in terms of reducing the classification error.

4.6.3 Performance comparison between different MOEAs

To further test the effectiveness of the proposed representation, three additional experiments were conducted. In the first experiment, the differential evolution crossover operator (DE) instead of SBX in NSGA-II was implemented. In a second and third experi-

ment, MOEA/D and SMS-EMOA were implemented as the searching techniques. Using MOEA/D and SMS-EMOA as the optimization algorithms would show the adaptability of the proposed CGP-based solution representation to different evolutionary searching strategies.

The experiments were run 10 times on the CIFAR-100 dataset. The DE operator was configured with CR = 1 and F = 0.5, while MOEA/D and SMS-EMOA used the same crossover and mutation mechanisms of NSGA-II and the configurable parameter to define the number of neighborhoods was set equal 4. The rest of the parameters were configured the same as those listed in Table 4.2.

Table 4.5: Comparison of CGP-NAS versions on the CIFAR-100 dataset, considering both methods for selecting solutions from the Pareto front. Classification error rate, the parameters and MAdds are expressed in millions (1×10^6).

	Knee			Best		
	Error rate (%)	Parameters	MAdds	Error rate (%)	Parameters	MAdds
CGP-NASV2 **NSGA-II-DE**	24.75 (27.15 ± 1.85)	1.09 (0.78 + 0.21)	86.49 (71.06+ 12.03)	21.02 (22.66± 0.99)	5.99 (5.89 + 2.75)	960.01 (1164.75 + 559.59)
CGP-NASV2 **MOEA/D**	29.47 (32.46 ± 1.42)	0.32 (0.34 ± 0.07)	36.71 (30.75± 7.70)	21.12 (23.88 ± 1.7)	5.30 (3.98 ± 1.11)	1021.62 (601.18± 186.12)
CGP-NASV2 **NSGA-II-SBX**	**23.57** **(28.43± 2.20)**	0.49 (0.53 ±0.13)	66.66 (55.66 ±19.14)	**20.63** **(22.49 ± 1.04)**	5.99 (6.50 ± 1.7)	827 (850.74± 476.08)
CGP-NASV2 **SMS-EMOA**	25.38 (26.84 ± 1.68)	0.95 (0.54 ± 0.19)	98.74 (61.09± 17.84)	21.55 (22.82 ± 1.06)	12.19 (7.10 ± 3.21)	1320.17 (846.15± 261.03)

Table 4.5 shows the results for CGP-NASV2 with different evolutionary strategies after 10 executions on the CIFAR-100 dataset. Our experiments revealed several interesting behaviors. In the canonical version of CGP-NASV2 with NSGA-II and SBX , we consistently obtained the best classification errors. When using the DE operator, we saw similar results to CGP-NASV2, with a slightly lower average and standard deviation, indicating more consistent behavior. Finally, MOEA/D performed similarly to DE while producing solutions with significantly lower complexity, potentially due to its decomposition approach. In the case of SMS-EMOA, we see a very similar performance to NSGA-II-SBX. However, we can notice that on average, in SMS-EMOA, the solutions have a lower or equal error than NSGA-II-SBX. This may be due to the indicator-based approach of SMS-EMOA, but the best solution located is worse than our baseline.

To further illustrate these findings, we present in Figure4.11 the aggregated Pareto fronts for each method below. To obtain them, we concatenated the Pareto fronts from each run, ten in total, and we keep it with the non-dominated solution.

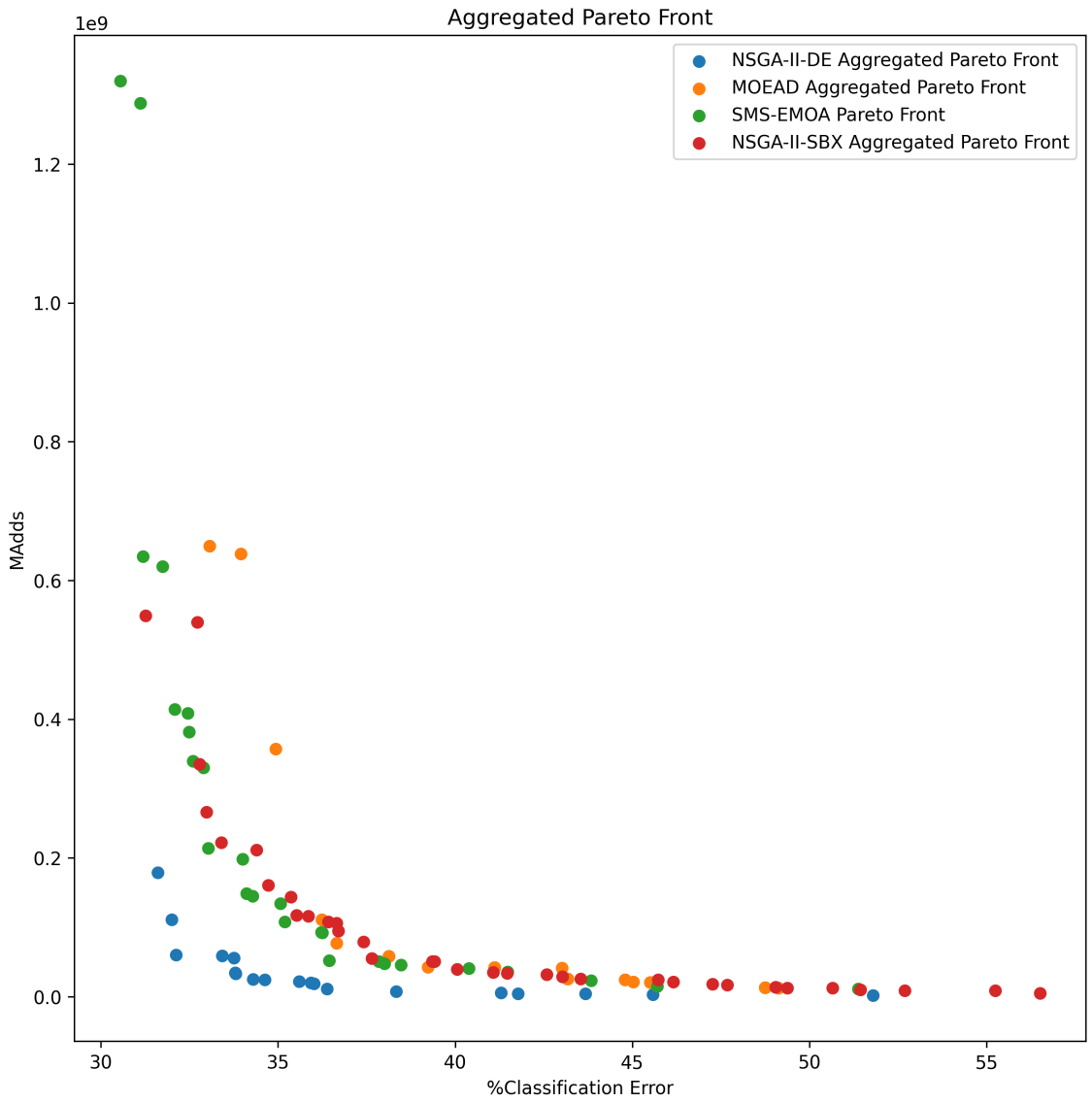


Figure 4.11: Aggregated Pareto Front of each evaluated Method on the CIFAR-100 Dataset. To obtain each Aggregated Pareto Front, the Pareto fronts from the 10 executions were used, and then the non-dominated solutions were selected, thus obtaining the aggregated Pareto front.

Analyzing Figure 4.11, we can observe different behaviors with the various MOEAs utilized. For example, we note that the NSGA-II-DE version with the differential evolution crossover operator stands out by providing solutions with notably lower complexity while maintaining good performance in terms of classification error. However, its distribution is slightly worse. Similarly, for MOEA/D, the solutions tend to have higher complexity but also higher error, reaffirming the trade-off that exists between both objectives. In the case of SMS-EMOA and NSGA-II-SBX, we observe very similar behaviors, excelling in a well-distributed set of solutions across the Pareto front.

To delve deeper and enable a more detailed analysis of the different algorithms, a comparison was performed using the hypervolume metric.

Ten runs were carried out for each MOEA, and all Pareto fronts were aggregated to obtain a single Pareto front. The Nadir point was taken as the reference point for evaluation. Using this reference point, each individual Pareto front was assessed to calculate the hypervolume value for each algorithm. The values obtained for each MOEA are shown in Table 4.6.

Table 4.6: Summary of Hypervolume Results.

Dataset	Mean ± STD
SMS-EMOA	0.4085 ± 0.0043
NSGA-II-DE	0.4007 ± 0.0135
MOEA/D	0.3671 ± 0.0057
NSGA-II-SBX	0.4051 ± 0.0052

Table 4.7: Pairwise Comparison of Hypervolume Results with NSGA-II-SBX as Baseline. (+: better, -: worse, =: no significant difference).

MOEA	Baseline: NSGA-II-SBX	Adjusted P-value	Result (+, -, =)
SMS-EMOA	NSGA-II-SBX	0.2058	=
NSGA-II-DE	NSGA-II-SBX	0.3527	=
MOEA/D	NSGA-II-SBX	0.0009	-

The Wilcoxon Rank-Sum Test was used to compare the hypervolume results of four MOEAs: SMS-EMOA, NSGA-II-DE, MOEA/D, and NSGA-II-SBX. This non-parametric test evaluates whether there are significant differences in the distributions of two independent

samples, making it appropriate for datasets that may not meet the assumptions of parametric tests. To address multiple pairwise comparisons, the Bonferroni correction was applied to the p-values to minimize the likelihood of Type I errors, as recommended in multiple hypothesis testing scenarios [83].

With these results, we can confirm that statistically, there is no significant difference between SMS-EMOA, NSGA-II-DE, and NSGA-II-SBX. However, MOEA/D is significantly worse than NSGA-II-SBX. Based on these findings and the results summarized in Table 4.5, we select the NSGA-II-SBX algorithm as the one delivering the best results, and it will therefore be used for subsequent experiments.

4.6.4 Comparison versus the state of the art

Tables 4.8 and 4.9 present a detailed comparison between the State of the Art works and the proposed algorithmic approaches CGP-NASV1 and CGP-NASV2 on the CIFAR-10 and CIFAR-100 datasets. A total of 4 human designs, 5 NAS single-objective, and 7 multi-objective proposals are considered for comparison.

From previous empirical assessments, it was determined that CGP-NASV2 provided the best overall results when compared to CGP-NASV1 and their baseline CGP-NAS. It was also analyzed that selecting the best solution from the Pareto front in terms of classification error would negatively affect the resulting evolved architecture in its complexity. Thus, a method to select trade-off solutions for both conflicting objectives, known as the knee and boundary methods, allowed the selection of an evolved architecture with more balanced performance metrics.

Another method to evaluate the performance of the proposed is by using the hypervolume metric. This metric assesses the distribution of solutions along the Pareto front, indicating how effectively they are spread across the objective function space. The hypervolume metric quantifies the total area covered by the Pareto front relative to a reference point, for which we use the nadir point (see section 11). This metric is computed after each generation, with higher values indicating better performance.

Figure 4.12 presents four box plots, each depicting the average hypervolume value per generation across 10 runs for the CIFAR-10 and CIFAR-100 datasets.

The final hypervolume for CGP-NASV1 on CIFAR-10 is **0.85(0.651 ± 0.147)**, and for CIFAR-100 it is **0.618(0.505 ± 0.08)**. Conversely, the results for CGP-NASV2 on CIFAR-10

Table 4.8: Comparison on CIFAR-10 dataset: Classification error rate, the number of parameters and Multiply-adds (MAdds) are expressed in millions (1×10^6), GPU-days and GPU Hardware.

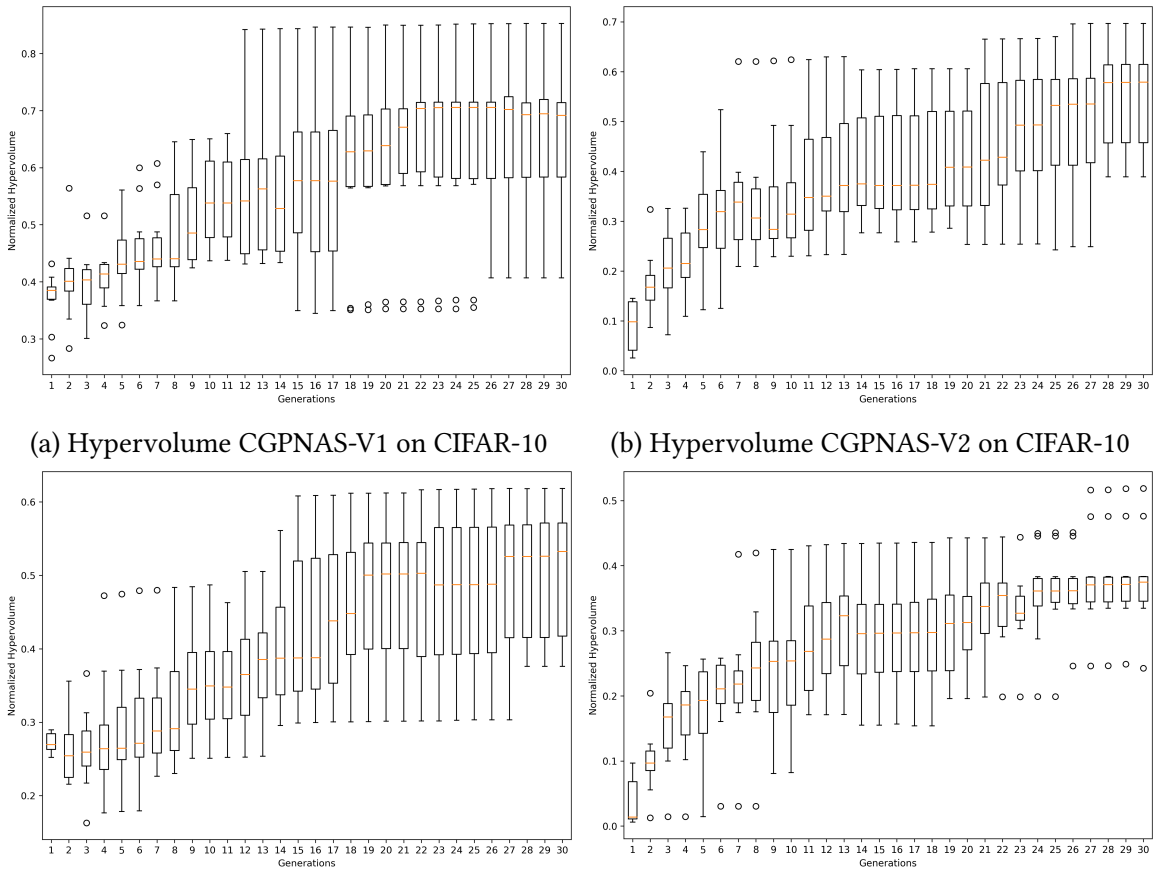
Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
DenseNet ($k = 12$) [84]	5.24	1.0	-	-	
ResNet ($depth = 101$) [72]	6.43	1.7	-	-	
ResNet ($depth = 1202$) [72]	7.93	10.2	-	-	
VGG [85]	6.66	20.04	-	-	
Single Objective Approaches					
CGP-CNN(ConvSet) [14]	5.92	1.50	-	8	Nvidia 1080Ti
CGP-CNN(ResSet) [14]	5.01	3.52	-	14.7	Nvidia 1080Ti
Large-Scale Evolution [86]	5.4	5.4	-	2750	-
AE-CNN [87]	4.3	2.0	-	27	Nvidia 1080 Ti
Genetic-CNN [16]	7.1	-	-	17	-
(Torabi et al., 2022) [59]	5.69	1.96	-	-	NVIDIA Tesla k80
Multi-Objective Approaches					
NSGANet [10]	3.85	3.3	1290	8	Nvidia 1080 Ti
NSGANetV1 [10]	4.67	0.2	-	27	Nvidia 2080 Ti
MOCNN [17]	4.49	-	-	24	Nvidia 1080 Ti
MOGIG-Net [65]	4.67	0.2	-	14	-
EEEA-Net [12]	2.46	3.6	-	0.52	Nvidia RTX 2080 Ti
EvoApproxNAS [67]	6.80	1.11	458.2	8.8	NVIDIA Tesla V100-SXM2
LF-MOGP [66]	4.13	1.07	-	10	NVIDIA GeForce 3090
CGP-NAS [11]	**4.86 (5.42 ± 0.46)**	**1.40**	**388.71**	**1.4**	**Nvidia Titan X**
CGP-NASV1-Best Solution	**4.23 (4.73 ± 0.44)**	**8.47 (8.74 ± 3.27)**	**1255.93 1161.09 ± 373.78)**	8.97	**Nvidia 1080Ti**
CGP-NASV1-Knee Solution	**5.66 (7.33 ± 1.55)**	**0.41 (8.74 ± 0.13)**	**70.13 (52.01 ± 21.09)**	8.97	**Nvidia 1080Ti**
CGP-NASV2-Best solution	**3.70 (4.07± 0.17)**	**4.04 (5.82 ± 2.70)**	**636.32 (818.61 ± 372.62)**	11.54	**Nvidia 1080Ti**
CGP-NASV2-Knee solution	**4.85 (5.59 ± 0.5)**	**0.78 (0.71 ± 0.31)**	**53.99 (79.44 ± 31.96)**	11.54	**Nvidia 1080Ti**

Table 4.9: Comparison on CIFAR-100 dataset: Classification error rate, the number of parameters and Multiply-adds (MAdds) are expressed in millions (1×10^6), GPU-days and GPU Hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
DenseNet ($k = 12$) [84]	24.42	1.0	-	-	
ResNet ($depth = 101$) [72]	25.16	1.7	-	-	
ResNet ($depth = 1202$) [72]	27.82	10.2	-	-	
VGG [85]	28.05	20.04	-	-	
Single Objective Approaches					
CGP-CNN(ConvSet) [14]	26.7	2.04	-	13	Nvidia 1080Ti
CGP-CNN(ResSet) [14]	25.1	3.43	-	10.9	Nvidia 1080Ti
Large-Scale Evolution [86]	23.0	40.4	-	2750	-
AE-CNN [87]	20.85	5.4	-	36	Nvidia 1080 Ti
Genetic-CNN [16]	29.03	-	-	17	-
(Torabi et al., 2022) [59]	26.03	2.56	-	-	NVIDIA Tesla V100-SXM2
Multi-Objective Approaches					
NSGANetV1 [10]	25.17	0.2	1290	27	Nvidia 2080 Ti
MOGIG-Net [65]	24.71	0.7	-	14	-
EEEA-Net [12]	15.02	3.6	-	0.52	Nvidia RTX 2080 Ti
LF-MOGP [66]	26.37	4.12	-	13	NVIDIA GeForce 3090
CGP-NAS [11]	**24.23 (26.41 ± 1.41)**	**5.43**	**1581**	**2.1**	**Nvidia Titan X**
CGP-NASV1 - Best Solution	**21.76 (24.20 ± 1.80)**	**3.60 (7.25 ± 3.24)**	**791.85 (792.02 ± 342.60)**	8.82	**Nvidia 1080Ti**
CGP-NASV1 - Knee Solution	**31.47 (33.34 ± 1.6)**	**1.06 (0.82 ± 1.12)**	**45.70 (30.81 ± 9.7)**	8.82	**Nvidia 1080Ti**
CGP-NASV2 - Best Solution	**20.63 (22.49 ± 1.04)**	**5.9 (6.50± 1.7)**	**827 (850.74 ± 476.08)**	11.28	**Nvidia 1080Ti**
CGP-NASV2 - Knee solution	**23.57 (28.43 ± 2.20)**	**0.49 (0.53 ± 0.13)**	**66.66 (55.66 ± 19.14)**	11.28	**Nvidia 1080Ti**

are **0.69(0.551 ± 0.10)**, and for CIFAR-100, it is **0.51(0.37 ± 0.07)**.

From these results and the previous figures, it can be observed that on the CIFAR-10 dataset, CGP-NASV1 exhibits more evenly distributed data, as indicated by the length of the whiskers and the size of the boxes, which suggests a higher degree of variability in the data across experiments. In contrast, CGP-NASV2 shows less variability, leading to a more concentrated distribution. A similar trend is observed on the CIFAR-100 dataset, where CGP-NASV1 tends to have more evenly distributed data, while CGP-NASV2 demonstrates a more concentrated distribution and more consistent solutions between experiments. However, in the final generations, CGP-NASV2 generates outliers, possibly due to sudden and unpredictable changes in the Pareto front caused by the stochastic nature of the search algorithm and the additional hyperparameters influencing this behavior. In summary, CGP-NASV1 appears to favor exploration, while CGP-NASV2 tends to favor exploitation towards the end of the search processes.



(a) Hypervolume CGPNAS-V1 on CIFAR-10

(b) Hypervolume CGPNAS-V2 on CIFAR-10

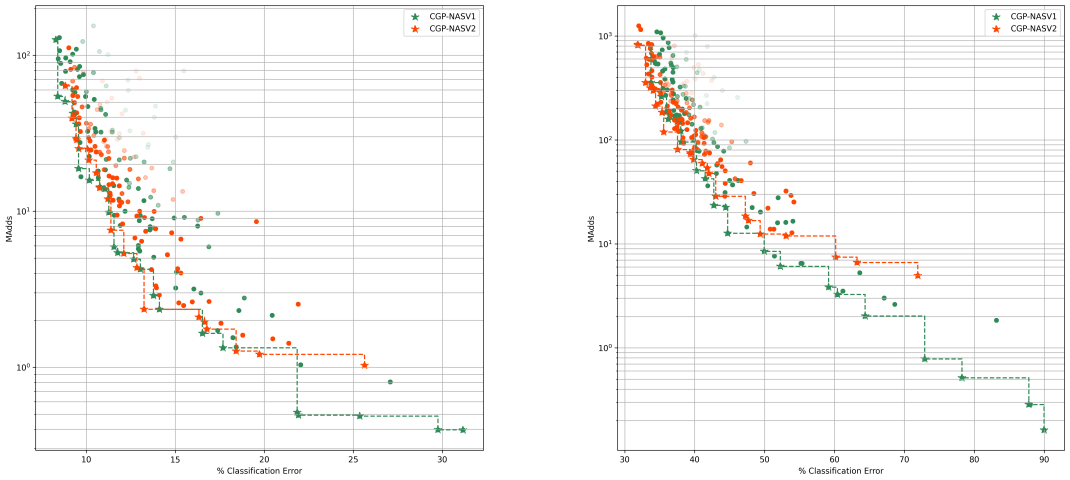
(c) Hypervolume CGPNAS-V1 on CIFAR-100

(d) Hypervolume CGPNAS-V2 on CIFAR-100

Figure 4.12: Box plot of normalized Hypervolume on CIFAR-10 and CIFAR-100.

The search process of CGP-NASV1 and CGP-NASV2 on the CIFAR-10 and CIFAR-100 datasets is illustrated in Figures 4.13a and 4.13b, respectively. The transparency of the circular marks in the figures indicates the relative early stages of the generations. The Pareto front for each approach is depicted by the corresponding lines. Initially, it is observed that the solutions are primarily located in the upper left zone, indicating more complex solutions. As the generations progress, solutions are placed further to the right. The region with the most optimal trade-off is located in the lower left corner.

From the figures, it is also apparent that the Pareto front for CGP-NASV1 tends to be more dispersed in the objective function space. Conversely, CGP-NASV2 demonstrates a tendency to focus on a more restricted area of the objective function space, typically leading to improved initial solutions compared to CGP-NASV1.



(a) CIFAR-10 Pareto front

(b) CIFAR-100 Pareto front

Figure 4.13: Comparison of the Pareto fronts between CGP-NASV1 and CGP-NASV2, and the population through generations.

Compared to other methods, particularly those designed by humans, our proposal demonstrates superior performance in terms of both classification error and the number of parameters on both datasets. When compared with single-objective methods, our proposal outperforms them in terms of classification error while showing a significant reduction in the number of parameters. It is important to note, however, that minimizing the number of parameters is not the primary objective of our proposal. When comparing with multi-objective methods in terms of classification error, the EEEA-NET [12] method shows better performance, albeit at the cost of a higher number of parameters. In other metrics, our proposal outperforms the other methods presented. Our proposal aims to find solutions with a favorable trade-off between classification error and MAdds, resulting in architectures with reduced parameters and MAdds.

In comparison with other proposals that use CGP as a method for architecture representation, such as CGP-CNN [14], Torabi [59], EvoApproxNas [67], and LF-MOGP [66], our proposal demonstrates superior performance. Works like EvoApproxNAS [67] focus on optimizing values at the hardware level with a multi-objective approach, keeping the CGP representation intact and adding some block functions like bottleneck and residual inverted to the function set while maintaining the CGP properties unchanged by only using mutations. On the other hand, none of these proposals modify CGP at the represen-

tation level, instead addressing specific problems, for example by proposing new operators (Torabi [59]) or mechanisms to improve the search process (LF-MOGP [66]), which seem to be important components since the results shown improve compared to other proposals, even though CGP-NASV2 shows superior performance with canonical MOEAs.

To demonstrate the applicability of CGP-NASV1 and CGP-NASV2 to other image classification tasks, we evaluated them on the SVHN dataset, with results reported in Table 4.10. When compared to human-designed methods, CGP-NASV1 and CGP-NASV2 achieve classification errors comparable to these methods, notably excelling in solutions with a smaller number of parameters. Our results surpass the multi-objective EvoApproxNAS [67] and the single-objective Bakhshi's [15] proposal in terms of classification error, number of parameters, and complexity. Particularly, CGP-NASV2, where parameters are included in the search, demonstrates better performance regarding the number of parameters and MAdds without negatively impacting the error rate.

Table 4.10: Comparison on SVHN dataset: Classification error rate, number of parameters, and Multiply-adds (MAdds) are expressed in millions (1×10^6), GPU-days, and GPU hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
FractalNet [88]	2.01	38.6	-	-	
Wide ResNet [89]	1.64	2.7	-	-	
ResNet ($depth = 101$) [72]	2.01	1.7	-	-	
DenseNet ($k = 24$) [84]	1.72	15.3	-	-	
Single Objective Approaches					
(Bakhshi et al., 2020) [15]	4.43	19	-	6	
Multi-Objective Approaches					
EvoApproxNAS [67]	3.09	0.90	247.3	8.8	NVIDIA Tesla V100-SXM2
CGP-NASV2-Best solution	**2.70 (2.87± 0.15)**	**2.21 (4.26 ± 1.88)**	**399.52 (697.71 ± 210.51)**	16.25	**Nvidia 1080Ti**
CGP-NASV2-Knee solution	**2.88 (3.05 ± 0.18)**	**0.49 (0.51 ± 0.16)**	**55.93 (49.31 ± 11.26)**	16.25	**Nvidia 1080Ti**
CGP-NASV1-Best solution	**3.03 (3.27± 0.33)**	**3.24 (5.02 ± 2.16)**	**556.58 (698.24 ± 381.22)**	13.87	**Nvidia 1080Ti**
CGP-NASV1-Knee solution	**3.77 (4.07± 0.42)**	**0.34 (0.28 ± 0.12)**	**24.22 (25.95 ± 7.68)**	13.87	**Nvidia 1080Ti**

CGP is known to be a robust method for architecture representation, and the use of real-based representation in our proposal improves performance. This can be attributed to the relaxation of the search space and the better utilization of MOEAs, which are designed to operate in a continuous domain.

In summary, CGP-NASV1 and CGP-NASV2 demonstrate the ability to identify solu-

tions with a favorable trade-off between two objectives, specifically by achieving a lower number of MAdds compared to other methods. Depending on the specific application or task at hand, different methods for selecting solutions, such as those presented in this work, may be utilized to determine the most appropriate solution for the given scenario.

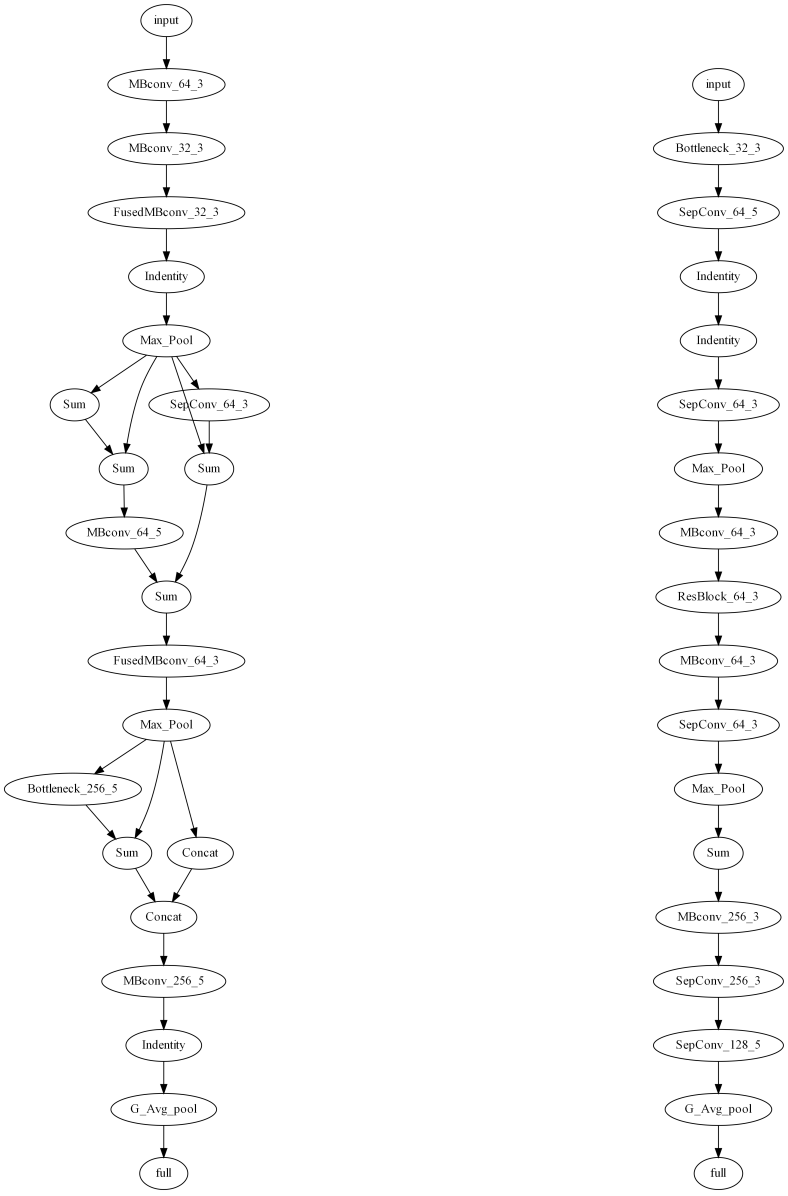
4.6.5 Evolved architectures

Figures 4.14 showcase two CGP-NASV2 evolved architectures selected through the knee and boundary method. Figure 4.14a, corresponding to the CIFAR-10 dataset, reveals an architecture with numerous branches and parameter-free blocks such as summation and concatenation. In contrast, Figure 4.14b for the CIFAR-100 dataset shows a more linear structure with complex blocks such as MBConv and SepConv. This indicates that CGP-NASV2 adapts to the more challenging CIFAR-100 dataset by selecting necessary, albeit complex, blocks without significantly inflating the number of parameters or MAdds, thus maintaining a low classification error.

4.7 Discussion

In this chapter we introduced a robust representation for multi-objective evolutionary approach for NAS applied to image classification tasks using the CIFAR-10,CIFAR-100 and SVHN datasets. The proposed methods, CGP-NASV1 and CGP-NASV2, excel at discovering CNN architectures that strike a balance between classification error and complexity, as measured by the number of MAdds. A notable strength of CGP is its versatility in exploring various evolutionary search strategies and operators.

Moreover, leveraging the block-chained representation in combination with continuous CGP provides significant advantages for evolutionary algorithms in NAS. This constraint effectively reduces the search space's dimensionality, offering enhanced control over layer connections within the architecture. This block-chained approach has gained traction in NAS research [1, 7], resulting in improved optimization efficiency. Our experimental results highlight that the block-chained constraint achieves competitive outcomes, outperforming our previous proposal [11], where this constraint was absent. The integration of block-chain encoding has markedly boosted the efficacy of our multi-objective NAS variants, aligning them closely with state-of-the-art methods in the field.



(a) For CIFAR-10 dataset (b) For CIFAR-100 dataset

Figure 4.14: CGP-NASV2 evolved CNN architectures selected by the knee and boundary method.

The findings confirm that CGP-NASV2 is effective in identifying CNN architectures with fewer parameters and MAdds compared to other leading methods. Additionally, our exploratory study examined the impact of different multi-objective evolutionary search strategies and operations. By substituting the crossover operator with the DE operator and employing other MOEAs like MOEA/D and SMS-EMOA, we observed multiple benefits from our CGP-based representation. These include accommodating variable-length architectures and adapting to various operators and search schemes due to our representation's operation in a real domain. Overall, this approach underscores the effectiveness and adaptability of the CGP-based representation for NAS.

Chapter 5

Progressive Self-Supervised Multi-objective NAS

Self-supervised learning aims to enable architectures to learn high-level representations from data using synthetic labels generated without human supervision [53]. Under this approach, an architecture, such as a CNN, is trained to minimize loss on pretext tasks (e.g., learning to rotate an image). The trained architecture then serves as a starting point (pretrained model) for supervised tasks with available labels (the downstream task). The primary advantage of these models is their ability to perform well across multiple tasks, making this approach highly attractive for NAS, as the costly search for a model becomes worthwhile if the model is applicable to numerous downstream tasks.

In this chapter, we will present a novel search strategy, as well as a variant of the performance estimation strategy. Thus focusing on two important parts of NAS. The search space used will be the one presented in Chapter 4, specifically CGP-NASV2.

Our model targets a self-supervised learning objective based on RotNet [56]. The idea is to empower our model to discover generic architectures without requiring labeled data, making them adaptable to other datasets and tasks. These architectures are integrated into a multi-objective searching algorithm that provides a spectrum of solutions where complexity and classification error are critical considerations. It is expected that by joining these two proposals, we will find more specialized architectures at each stage of the search, this thanks to a progressive search, on the other hand we seek to reduce the time thanks to the self-supervised learning, because a reduction in time is expected when we apply transfer learning to other tasks or datasets.

5.1 Progressive search - Search strategy

As detailed in Chapter 4, CGP-NASV2 employs high-level blocks for solution representation, with each block representing a segment of the CNN architecture. Figure 5.1 illustrates an architecture comprising three blocks. The general idea is to divide the search process into N different stages, corresponding to the number of "Normal" blocks in the representation used and the established number of generations, as depicted in Figure 5.2. The search is focused on the current block by applying crossover and mutation operators solely to this block, as these are the operators that generate modifications in the resulting descendants, see Fig 5.3.

The other blocks are not considered and are substituted with the ConvBlock function to simplify the architecture, ensuring that these blocks do not interfere. As we move to the next stage, the previous blocks are fixed, meaning they can no longer be modified in the future, and will only connect with the current blocks.

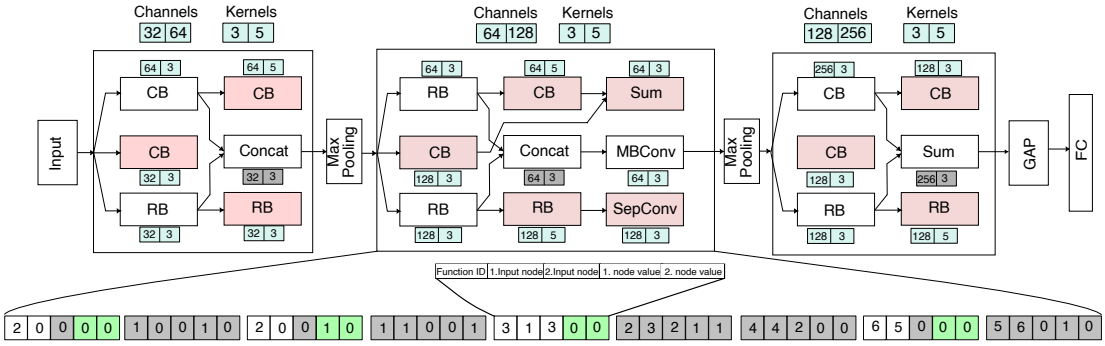


Figure 5.1: CGP-NASV2 block-chained representation with the hyperparameters directly encoded.

The MOEA modified with this new crossover and mutation mechanism was NSGA-II, as mentioned in Chapter 4 Section 4.3, which yielded the best results. It is important to note that the modification was only applied to this stage of the algorithm.

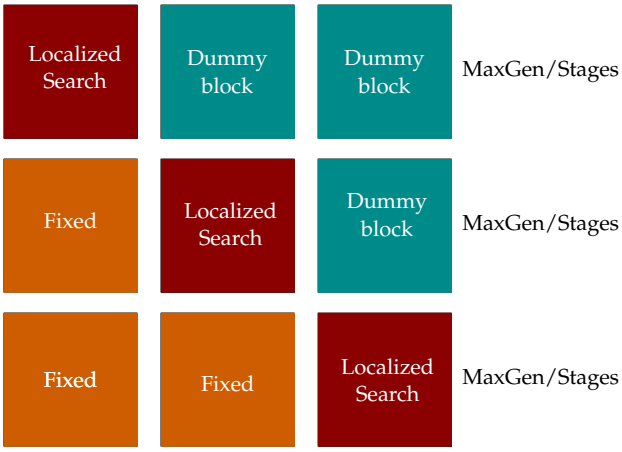


Figure 5.2: Proposed progressive search scheme.

5.2 Self-supervised evaluation - Performance estimation strategy

The evaluation utilizes a self-supervised approach based on the RotNet [56] model with the CIFAR-100 dataset. We generated a new dataset by replacing the original CIFAR-100 training set classes with rotation levels, resulting in four distinct classes (0°, 90°, 180°, 270°) as shown in Figure 5.4 instead of the original 100 classes. This data is then used to classify rotations, enabling us to train a model on a related task without the need for extensive labeled data. Once a competitive architecture is identified, it is further trained with labeled data for image classification. It is important to note that an architecture learned from this dataset can also be applied to other downstream tasks. Therefore, with this approach, the complexity when evaluating the architecture is reduced in a certain way, because the number of parameters in the final layer of the architecture is reduced, since it is related to the number of classes.

To evaluate the fitness of each solution, all blocks are assembled together, and the resulting architecture is trained using the modified CIFAR-100 dataset, thereby obtaining the classification error. The complexity of the model is measured by summing all the addition and multiplication operations within the architecture, resulting in the total number of MAdds.

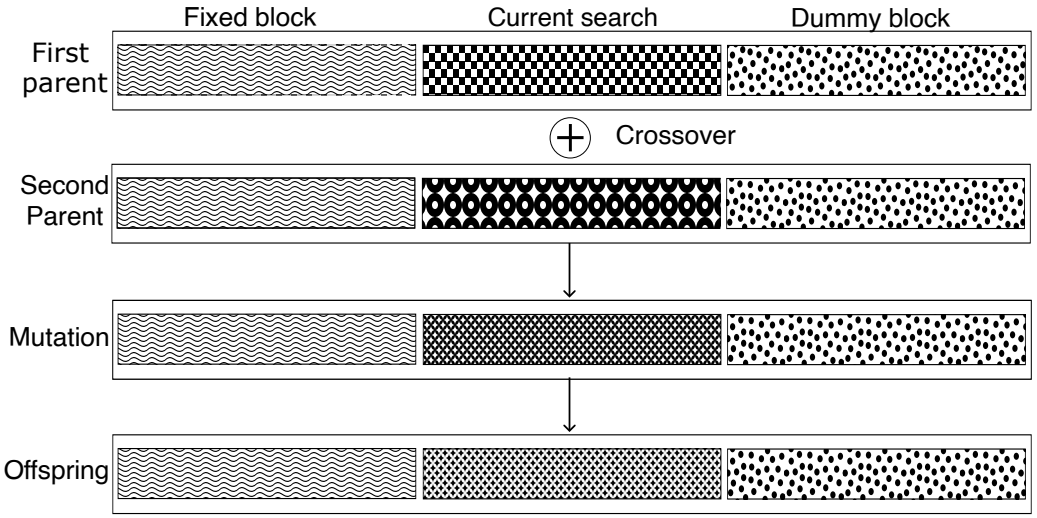


Figure 5.3: Example of how the mating stage is carried out, the block where the current search is performed, changes after each stage defined by a number N of generations.

5.3 Experimental framework

The proposed experimental framework first assesses the performance of the best-evolved architectures in terms of accuracy. After that, a multiple-criteria decision analysis is applied to those models that achieved a good trade-off performance in terms of both accuracy and complexity (measured in MAdds). The experimental settings and benchmark datasets are defined next.

Table 5.1: Experimental Settings.

Parameter	Value
CGP rows and columns	10×4
Mutation probability	$Pm = 0.3$
Crossover probability	$Pc = 0.9$
Population	16
Generations	30
CNN train epochs	36

The search was performed on the modified CIFAR-100 dataset. For self-supervised learning, only the training partition of the modified CIFAR-100 was used at this stage: 80% for training and 20% for validation of the evaluated architectures during the evolu-

(a) 0° rotation (b) 90° rotation (c) 180° rotation (d) 270° rotation

Figure 5.4: Images were rotated by random multiples of 90 degrees (i.e., 0°, 90°, 180°, or 270°).

tionary search. The evolved architectures were then transferred to the CIFAR-100, CIFAR-10, and SVHN datasets previously defined in the chapter 4 , and CINIC-10 [90] dataset. The CINIC-10 dataset contains 270,000 color images divided into 10 categories, including airplanes, cars, birds, and animals, similar to CIFAR-10. The dataset was created by combining CIFAR-10 images with additional images sourced from the ImageNet dataset, providing a more diverse and larger dataset compared to CIFAR-10.

We followed the training mechanisms for the evolved CNN architecture according to CGP-NASV2: Stochastic Gradient Descent (SGD) as the optimizer together with the cosine annealing learning rate schedule. Our initial learning rate was configured at 0.025, while the momentum was set to 0.9, and the weight decay to 0.0005. The batch size was established at 128, and we conducted a total of 36 training epochs during the evolutionary search.

For both training and testing data, we implemented the following preprocessing steps: a 4-pixel mean subtraction padding on each side and random cropping with a 32×32 patch or its horizontally flipped counterpart. To enhance the training process, we incorporated an auxiliary head classifier [9, 29], which is concatenated after the second reduction block. The loss from this auxiliary head classifier was scaled by a constant factor of 0.4 and added to the loss of the original architecture. This step is applied when we retrain the architectures with a complete dataset. In the final stages, the selected solutions are trained for 600 epochs with the application of the cutout preprocessing technique and a batch size set to 96.

All experiments were conducted on a supercomputing node equipped with 2 Intel

Xeon E5-2650 v4 @ 2.20GHz processors, 6 Nvidia GTX 1080 Ti GPU cards, and 128GB of RAM. The system operated on the CentOS 7 OS.

5.4 Experimental results

Our empirical assessment first involves a general performance comparison against state-of-the-art approaches. Subsequently, we analyze the generalization achieved by the self-supervised approach using the Grad-CAM [91] method.

5.4.1 Comparison with state-of-the-art

Tables 5.2 - 5.5 present a comparison of the results between the state-of-the-art methods and our proposed approach on the CIFAR-100, CIFAR-10, SVHN, and CINIC-10 datasets. As detailed in Section 4, the search was conducted on the modified CIFAR-100 dataset. The total search time was **5.79** GPU days on the modified CIFAR-100 dataset. It is important to note that the search was not performed on the aforementioned datasets; they were only used for transfer learning.

We conducted ten individual experiments on the modified CIFAR-100 dataset. We report the best solution, average, and standard deviation, and present two types of solutions: the “Best Solution”, which refers to the best-evolved solution found in 10 executions, and the “Knee Solution," determined by the Knee and Boundary Selection Method [82], which identifies the solution closest to the ideal point by balancing the two objectives.

These datasets were selected because they are the most commonly used in the NAS domain, starting with CIFAR-100 (see Table 4.9). Our proposal demonstrates competitive performance with state-of-the-art solutions, including both human-designed and single-objective approaches. In multi-objective proposals, the ability to find solutions with a good trade-off between the number of parameters and low classification error is highlighted. The proposed approach shows performance very similar to CGP-NASV2 [29], with the added advantage of reduced time in terms of GPU-days. When compared with other multi-objective methods, the EEEA-NET [12] method exhibits better performance in terms of classification error. However, in terms of the number of parameters and MAdds, our proposal outperforms most other methods presented. Our approach aims to find solutions with a favorable trade-off between classification error and MAdds. Compared to other

Table 5.2: Comparison on CIFAR-100 dataset: Classification error rate, the number of parameters and MAdds are expressed in millions (1×10^6), GPU-days, and GPU Hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
DenseNet ($k = 12$) [84]	24.42	1.0	-	-	
ResNet ($depth = 101$) [72]	25.16	1.7	-	-	
ResNet ($depth = 1202$) [72]	27.82	10.2	-	-	
VGG [85]	28.05	20.04	-	-	
Single Objective Approaches					
CGP-CNN(ConvSet) [14]	26.7	2.04	-	13	Nvidia 1080Ti
CGP-CNN(ResSet) [14]	25.1	3.43	-	10.9	Nvidia 1080Ti
Large-Scale Evolution [86]	23.0	40.4	-	2750	-
AE-CNN [87]	20.85	5.4	-	36	Nvidia 1080 Ti
Genetic-CNN [16]	29.03	-	-	17	-
(Torabi et al., 2022) [59]	26.03	2.56	-	-	NVIDIA Tesla V100-SXM2
Multi-Objective Approaches					
NSGANetV1 [10]	25.17	0.2	1290	27	Nvidia 2080 Ti
MOGIG-Net [65]	24.71	0.7	-	14	-
EEEA-Net [12]	15.02	3.6	-	0.52	Nvidia RTX 2080 Ti
LF-MOGP [66]	26.37	4.12	-	13	NVIDIA GeForce 3090
CGP-NAS [11]	24.23 (26.41 ± 1.41)	5.43	1581	2.1	Nvidia Titan X
CGP-NASV2 - Best solution	21.18 (22.55 ± 1.24)	4.02 (4.43 ± 1.57)	457.55 (559.60 ± 476.08)	6.25	Nvidia 1080Ti
CGP-NASV2 - Knee solution	26.35 (29.19 ± 2.20)	0.69 (0.46 ± 0.14)	54.09 (46.96 ± 16.50)	6.25	Nvidia 1080Ti
Progressive - Best solution	22.76 (25.63 ± 2.44)	2.89 (2.67 ± 1.34)	629.11 (545.30 ± 264.25)	5.79	Nvidia 1080Ti
Progressive - Knee solutions	26.57 (30.45 ± 2.67)	0.53 (0.43 ± 0.22)	66.39 (51.89 ± 28.89)	5.79	Nvidia 1080Ti

Table 5.3: Comparison on CIFAR-10 dataset: Classification error rate, the number of parameters and MAdds are expressed in millions (1×10^6), GPU-days and GPU Hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
DenseNet ($k = 12$) [84]	5.24	1.0	-	-	
ResNet ($depth = 101$) [72]	6.43	1.7	-	-	
ResNet ($depth = 1202$) [72]	7.93	10.2	-	-	
VGG [85]	6.66	20.04	-	-	
Single Objective Approaches					
CGP-CNN(ConvSet) [14]	5.92	1.50	-	8	Nvidia 1080Ti
CGP-CNN(ResSet) [14]	5.01	3.52	-	14.7	Nvidia 1080Ti
Large-Scale Evolution [86]	5.4	5.4	-	2750	-
AE-CNN [87]	4.3	2.0	-	27	Nvidia 1080 Ti
Genetic-CNN [16]	7.1	-	-	17	-
(Torabi et al., 2022) [59]	5.69	1.96	-	-	NVIDIA Tesla k80
Multi-Objective Approaches					
NSGANet [10]	3.85	3.3	1290	8	Nvidia 1080 Ti
NSGANetV1 [10]	4.67	0.2	-	27	Nvidia 2080 Ti
MOCNN [17]	4.49	-	-	24	Nvidia 1080 Ti
MOGIG-Net [65]	4.67	0.2	-	14	-
EEEA-Net [12]	2.46	3.6	-	0.52	Nvidia RTX 2080 Ti
EvoApproxNAS [67]	6.80	1.11	458.2	8.8	NVIDIA Tesla V100-SXM2
LF-MOGP [66]	4.13	1.07	-	10	NVIDIA GeForce 3090
CGP-NAS [11]	4.86 (5.42 ± 0.46)	1.40	388.71	1.4	Nvidia Titan X
CGP-NASV2-Best solution	3.70 (4.07± 0.17)	4.04 (5.82 ± 2.70)	636.32 (818.61 ± 372.62)	11.54	Nvidia 1080Ti
CGP-NASV2-Knee solution	4.85 (5.59 ± 0.5)	0.78 (0.71 ± 0.31)	53.99 (79.44 ± 31.96)	11.54	Nvidia 1080Ti
Progressive - Best solution	4.53 (5.30 ± 0.9)	2.11 (2.65 ± 1.34)	684.52 (545.21 ± 264.25)	5.79	Nvidia 1080Ti
Progressive - Knee solutions	5.38 (7.43 ± 1.42)	0.82 (0.41 ± 0.22)	87.30 (51.80± 28.88)	5.79	Nvidia 1080Ti

CGP-based architecture representation methods, such as CGP-CNN [14], Torabi [59], and LF-MOGP [66], our proposal shows superior performance.

In Table 5.3, a comprehensive comparison of various methods on the CIFAR-10 dataset is presented. Our method outperforms single-objective proposals designed by humans in terms of both parameters and classification error. Comparable results to those achieved on the CIFAR-100 dataset were observed when compared to other multi-objective proposals. It's noteworthy that our approach did not conduct direct search within the dataset but instead transferred discovered architectures using self-supervised learning.

Finally, we evaluated the proposed approach on the SVHN and CINIC-10 datasets, and the corresponding results are shown in Tables 5.4 and 5.5. When compared with methods designed by humans, our method achieves very close performance values; however, the solutions identified by our approach demonstrate a significant reduction in terms of pa-

rameters. In comparison to CGP-NASV2, it is evident that, on average, our solutions are less complex with reduced parameter counts.

On the CINIC-10 dataset, which is more demanding, our method again shows competitive performance comparable to human-designed methods, highlighting our approach for delivering low-complexity solutions compared to other methods. We have demonstrated that employing a progressive search combined with self-supervised learning leads to the discovery of more generalizable architectures.

It is important to emphasize that our search process did not directly use the original dataset labels. This achievement is significant as our method adapts evolved architectures through self-supervised learning. This simplifies the task because, unlike conventional methods that use all 100 labels for the CIFAR-100 dataset, our approach utilizes only 4 labels corresponding to the rotation levels (0°, 90°, 180°, 270°) mentioned earlier.

Table 5.4: Comparison on the SVHN dataset : Classification error rate, number of parameters and MAdds are expressed in millions (1×10^6), GPU-days and GPU Hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
FractalNet [88]	2.01	38.6	-	-	
Wide ResNet [89]	1.64	2.7	-	-	
ResNet ($depth = 101$) [72]	2.01	1.7	-	-	
DenseNet ($k = 24$) [84]	1.72	15.3	-	-	
Single Objective Approaches					
(Bakhshi et al., 2020) [15]	4.43	19	-	6	
Multi-Objective Approaches					
EvoApproxNAS [67]	3.09	0.90	247.3	8.8	NVIDIA Tesla V100-SXM2
CGP-NASV2 - Best solution	2.70 (2.87± 0.15)	2.21 (4.26 ± 1.88)	399.52 (697.71 ± 210.51)	16.25	Nvidia 1080Ti
CGP-NASV2 - Knee solution	2.88 (3.05 ± 0.18)	0.49 (0.51 ± 0.16)	55.93 (49.31 ± 11.26)	16.25	Nvidia 1080Ti
Progressive - Best Solution	2.95 (3.32± 0.43)	2.87 (2.65 ± 1.34)	629.02 (545.21 ± 264.25)	5.79	Nvidia 1080Ti
Progressive - Knee Solution	2.82 (3.64± 0.49)	0.51 (0.41 ± 0.22)	66.29 (51.80 ± 28.88)	5.79	Nvidia 1080Ti

5.4.2 Visual analysis of the evolved architectures

In this section, we conduct a visual analysis of the architecture discovered by our proposed method using Grad-CAM [91]. Figure 5.5 presents the architecture trained with the self-supervised approach. Different layers of the network are examined to observe the activated regions within images. Our initial hypothesis suggests that employing the

Table 5.5: Comparison on the CINIC-10 dataset: Classification error rate, number of parameters and MAdds are expressed in millions (1×10^6), GPU-days and GPU Hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
VGG-16 [92]	12.23	14.7	-	-	
ResNet-18 [92]	9.73	11.2	-	-	
MobileNet [92]	18.00	3.2	-	-	
DenseNet-121 [92]	8.74	7.0	-	-	
GoogLeNet [92]	8.83	6.2	-	-	
Multi-Objective Approaches					
Progressive - Best solution	12.16 (13.74 ± 1.4)	3.07 (2.65 ± 1.34)	584.71 (545.21 ± 264.25)	5.79	Nvidia 1080Ti
Progressive - Knee solutions	14.33 (17.00 ± 1.92)	0.51 (0.41 ± 0.22)	66.29 (51.80± 28.88)	5.79	Nvidia 1080Ti

self-supervised approach leads the CNN architecture to focus on high-level features in images, such as circular or square shapes, as well as more complex features like heads or eyes. This specific architecture was selected from those discovered by our method.

The evolved architecture underwent training in two distinct manners: initially in a self-supervised manner using the modified CIFAR-100 dataset, and subsequently in a supervised manner using the CINIC-10 dataset.

Two random images from the CINIC-10 dataset are depicted in Figures 5.6 and 5.7. The first row displays activation maps from the self-supervised training, while the second row shows those from the supervised training. Each image corresponds to activation maps extracted at different depths of the architecture, indicated by the respective layer.

Upon observation, the architecture discovered through the self-supervised approach initially highlights more generalized details and sharper edges within the input images. Conversely, when transferred and trained in a supervised manner, the activation maps from early layers tend to emphasize regions relevant to the classification task (see Fig. 5.6). For instance, as illustrated in Fig. 5.7, the area corresponding to the bird's head prominently influences the final decision. In contrast, in the self-supervised approach, the shape and distinct edges of the bird's wings appear more significant.

5.5 Discussion

In this Chapter based on the results obtained, we found that the architectures discovered by our method prioritize operations conducive to enhanced generalization, as evidenced by the Grad-CAM analysis. Moreover, our approach demonstrates promising outcomes alongside a significant reduction in GPU Days. The utilization of self-supervised learning, particularly through techniques like RotNet which estimate geometric transformations, enables networks to focus on object shapes, edges, and textures in images. This approach facilitates learning of crucial image features. Consequently, the generalization achieved allows for the transfer of discovered architectures to datasets with similar characteristics. In summary, this approach establishes a novel method for neural architecture search, leveraging self-supervised learning as a performance estimation strategy to develop models tailored to user-defined specifications in data-limited environments.

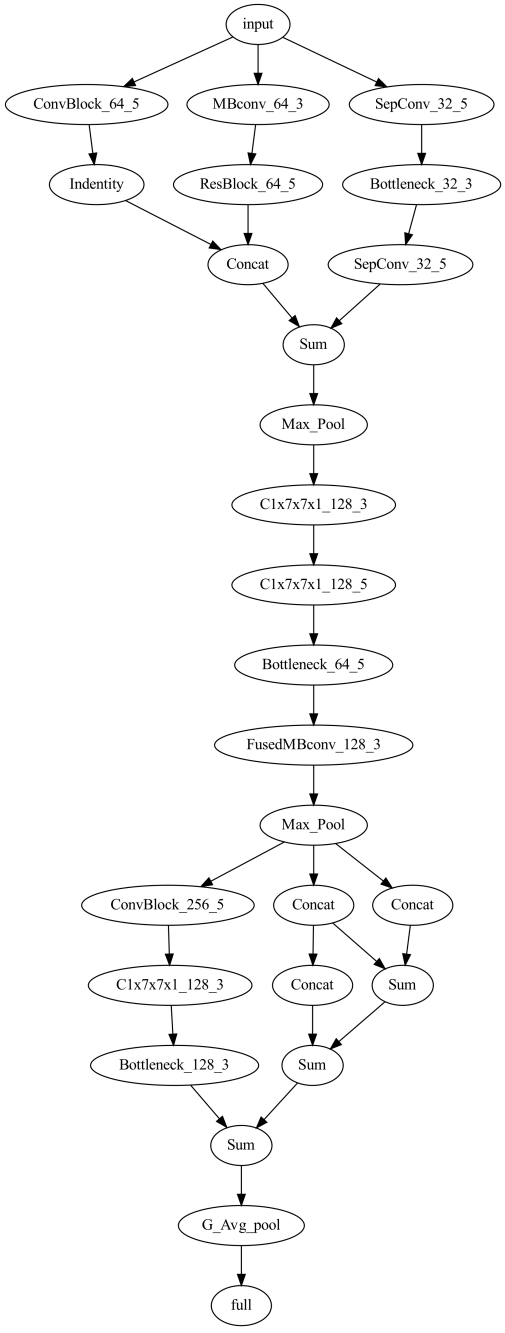
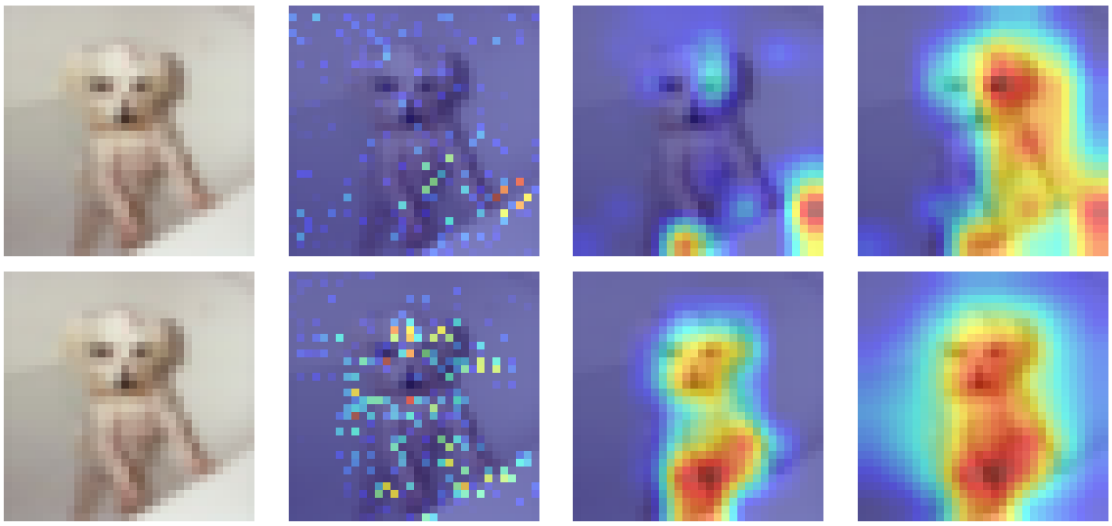
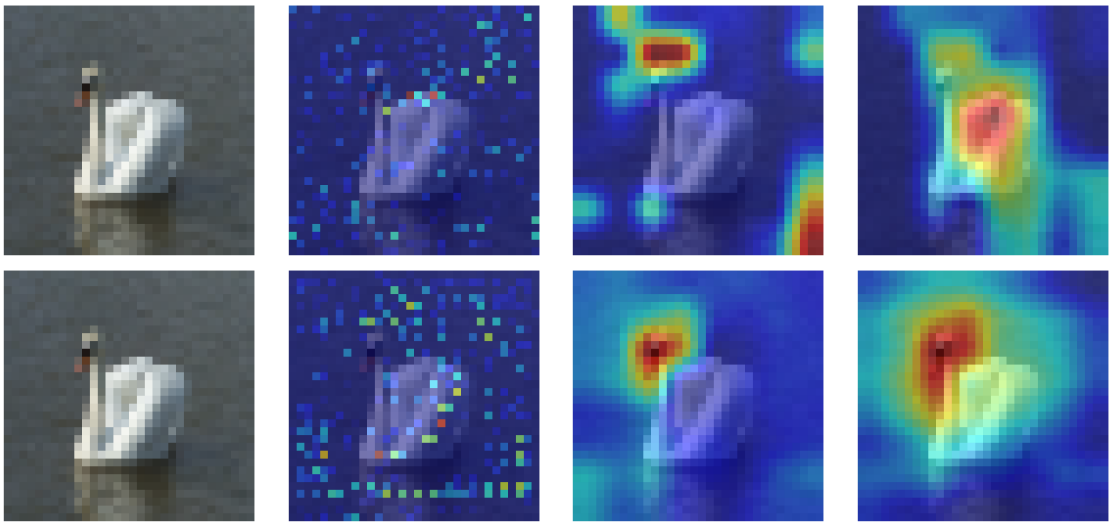


Figure 5.5: Visualization of the evolved architecture, comparing both self-supervised and supervised approaches using Grad-CAM.



(a) Input image (b) Layer 7 (c) Layer 17 (d) Layer 23

Figure 5.6: Activation maps extracted using Grad-CAM. Top: architecture learned with self-supervised learning. Bottom: architecture learned with supervised learning.



(a) Input Image (b) Layer 7 (c) Layer 17 (d) Layer 23

Figure 5.7: Activation maps extracted using Grad-CAM

Chapter 6

Incremental Training Dataset Expansion

In this chapter, we introduce a new, simple yet effective strategy to estimate the performance of the search for neural architectures. This new strategy derives from the lower fidelity estimates explained in Section 2.1.3. Under this technique, when evaluating an architecture, a reduced amount of training data is used to estimate the model's accuracy. Our proposed strategy aims to use a linear increase in the size of the training set used to train each architecture, i.e., the training dataset increases as the number of generations increases. The search space and search strategy are based on the methods explained in Chapter 4, specifically CGP-NASV2. With this, it is expected to drastically reduce the total time of the search, since the smaller the amount of data used, the training of CNNs architectures in early generations will be performed quickly, also since the initial solutions are random, it should not focus the expenditure of resources on these solutions as they are likely to be discarded in the following generations.

6.1 The proposed performance estimation strategy

In this new approach, we focus on methods to speed up the performance estimation of architectures within the domain of lower-fidelity estimates. Typically, in this class of methods, a subset of the training set is employed to estimate the performance of each architecture during its evaluation. In various studies, 80% of the training subset is utilized, with the remaining 20% used for model validation, thus avoiding testing on the test partition of datasets [9, 11]. Such a choice obviously influences both the quality of the solutions

explored by the considered NAS approach and the GPU time required to find them.

Now, we instead consider changing the amount of data used for training dynamically during the NAS optimization process. More particularly, we propose to start from a small subset and progressively increase the size of the training set used to evaluate the evolved architectures. Specifically, depending on the total number of generations G available to evolve the corresponding solutions, the size of the training partition is gradually increased at each generation g, starting with $X\%$ and ending at $Y\%$ of the dataset size. More formally, we consider a simple linear increase, where at each current generation $g \in \{1, \ldots, G\}$ of the considered evolutionary search process, the size S of the training dataset is set to:

$$S = (Y - X) \cdot \frac{g - 1}{G - 1} + X \tag{6.1.1}$$

Therefore, the idea is to incrementally grow the dataset size with each generation, aiming to create an environment conducive to convergence in the final generations, where the dataset size is larger. In other words, we gradually use more training data as the optimization process advances, resulting in increased confidence in the estimated performance of the discovered architectures. As a consequence of this method, our objective is to reduce the total algorithm time, as there is a clear correlation between the size of the data used to evaluate the architecture and the required training time.

Although this idea of incrementally expanding the training dataset in the NAS setting is independent of the (evolutionary) optimization algorithm and framework, we use the CGP-NASV2 representation as the baseline. using CGP-NASV2 will allow us to observe the search dynamics inferred by our proposed approach from a multi-objective perspective. In our case, the considered objectives are to reduce classification error and the complexity of the architecture measured in MAdds.

In the remainder of this approach, we consider coupling the proposed approach with the well-established NSGA-II [35] as a search strategy. We use the standard SBX crossover and polynomial mutation, considered the default evolutionary operations in NSGA-II. Thanks to the flexibility offered by CGP-NASV2, the underlying representation employing real domain encoding is easily adaptable to be integrated into such standard MOEAs. Although other MOEAs could have been considered, we do not explore such alternatives since our goal is to focus on the impact of the proposed incremental training dataset expansion approach.

6.2 Experimental framework

In this section, we will define the experimental setup, detailing the configurations and parameters used. Then, we will present and analyze the results, followed by a comparison with other state-of-the-art methods, highlighting our approach's strengths and potential improvements.

Five runs were executed, and the following parameters (see Table 6.1) were used as recommended in previous work from the literature[29]. CGP-NASV2 was used as the baseline, and the same multiple-criteria decision analysis was applied to select the solution with the best trade-off performance in terms of both accuracy and complexity in MAdds. The CIFAR-100 dataset was used to evaluate our model. A total of 30 generations were used, with the percentage of the training set increasing from 20% to 80%.

Table 6.1: Parameters configuration.

Parameter	Value
CGP rows and columns	10×4
Generations (G)	30
Population	24
Mutation probability (P_m)	0.3
CNN training epochs	36
Crossover probability (P_c)	0.9
Initial dataset size (X)	20%
Final dataset size (Y)	80%

Notice that we keep the same training procedures as specified by CGP-NASV2 for the evolved CNN architecture. Stochastic Gradient Descent (SGD) served as the optimizer with a cosine-annealing learning rate schedule. Our initial learning rate was set to 0.025, with a momentum of 0.9 and weight decay of 0.0005. The batch size was fixed at 128, and a total of 36 training epochs were conducted during the evolutionary search.

For both the training and testing datasets, we applied specific preprocessing steps: a 4-pixel mean subtraction padding on each side and random cropping using a 32×32 patch or its horizontally flipped counterpart. To augment the training process, an auxiliary head classifier [9, 29] was integrated and concatenated after the second reduction block. The loss from this auxiliary head classifier varied, scaling by a constant factor of 0.4, and was

added to the loss of the original architecture. This step was implemented during the retraining of architectures using a complete dataset. In the final stages, the selected solutions underwent training for 600 epochs, employing the cutout preprocessing technique, with a batch size set to 96.

Our experiments were carried out on a supercomputing node featuring 2 Intel Xeon Gold 6126 processors, 2 Nvidia Tesla V100 (32 GiB) GPU cards, and 192GB of RAM. The baseline CGP-NASV2 results were obtained using eight Nvidia 1080Ti (16 GiB) GPU cards. Initially, this suggests a difference in processing time. However, our approach aims to use less information to achieve faster training. This implies that even if the same GPUs were used, the training time for our proposal should be shorter.

6.3 Results

To evaluate the performance of the proposed method, we consider both single-objective and multi-objective perspectives, as detailed below.

6.3.1 Overall performance of the best and knee Solutions

Initially, we analyze the relative performance of our approach compared to existing methods, including single-objective NAS approaches that focus solely on the objective f_1 initially stated in Equation (4.1.2) as the primary optimization problem. This allows us to appreciate the capability of our method to guide the search toward competitive architectures relative to other state-of-the-art single-objective approaches from the literature. However, since our approach is inherently multi-objective, the final output is a complete Pareto set approximation. Therefore, we use multiple-criteria decision-making tools to extract a single solution for comparison. For this purpose, we present two types of results: (i) the solution with the best classification error (ignoring the second MAdds objective) and (ii) the Knee solution, which is closest to the reference point in the Pareto approximation. As mentioned in CGP-NASV2, the Knee solution represents a reasonable trade-off between accuracy and complexity in the computed CNN architectures.

Table 6.2 provides an overall summary of the results, showing the performance of our proposal against the baseline CGP-NASV2 and other state-of-the-art approaches. The different approaches are divided into three categories: architectures designed by humans,

Table 6.2: Comparisons on CIFAR-100 dataset: Classification error rate, the number of parameters, and MAdds are expressed in millions (1×10^6), GPU-days, and GPU hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
VGG [85]	28.05	20.04	-	-	-
DenseNet ($k = 12$) [84]	24.42	1.0	-	-	-
ResNet ($depth = 1202$) [72]	27.82	10.2	-	-	-
ResNet ($depth = 101$) [72]	25.16	1.7	-	-	-
Single Objective Approaches					
AE-CNN [87]	20.85	5.4	-	36	Nvidia 1080 Ti
Genetic-CNN [16]	29.03	-	-	17	-
(Torabi et al., 2022) [59]	26.03	2.56	-	-	NVIDIA Tesla V100-SXM2
CGP-CNN(ConvSet) [14]	26.7	2.04	-	13	Nvidia 1080Ti
CGP-CNN(ResSet) [14]	25.1	3.43	-	10.9	Nvidia 1080Ti
Large-Scale Evolution [86]	23.0	40.4	-	2750	-
Multi-Objective Approaches					
NSGANetV1 [10]	25.17	0.2	1290	27	Nvidia 2080 Ti
EEEA-Net [12]	15.02	3.6	-	0.52	Nvidia RTX 2080 Ti
LF-MOGP [66]	26.37	4.12	-	13	NVIDIA GeForce 3090
MOGIG-Net [65]	24.71	0.7	-	14	-
CGP-NASV2 [29] - Knee solution	23.57 (28.53 ± 3.13)	0.49 (0.47 ± 0.09)	66.66 (51.07 ± 17.81)	12.15	Nvidia 1080Ti
CGP-NASV2 [29] - Best solution	20.63 (22.23 ± 1.46)	5.99 (6.75± 1.80)	827.14 (876.27± 344.87)	12.15	Nvidia 1080Ti
Performance estimation - Knee solutions	**28.23** (**30.18 ± 1.82**)	0.79 (0.24 ± 0.24)	**81.42** (**48.83± 19.03**)	2.62	Nvidia V100
Performance estimation - Best solution	**22.84** (**23.94 ± 1.02**)	5.90 (5.09 ± 1.15)	**716.00** (**716.70 ± 196.10**)	2.62	Nvidia V100

architectures designed by single-objective methods, and those designed by multi-objective methods. When comparing our approach to CGP-NASV2, with the primary objective of reducing the overall search time, we observe a reduction to 2.62 GPU days from the baseline's 12.15 GPU days, demonstrating that our approach is 4.6 times faster. Additionally, when comparing the results in terms of classification error and MAdds objectives, the differences in classification error are minimal. However, concerning complexity, our proposal achieves better outcomes on average. This becomes even clearer when we analyze the results from a multi-objective perspective, as detailed in the following paragraphs.

6.3.2 Overall Performance from a multi-objective perspective

For illustrative purposes, we present in Figure 6.1 the Pareto front approximations obtained by our method and the existing CGP-NASV2 baseline. These fronts are derived by aggregating the independent executions of both approaches and subsequently applying a filtering procedure to retain only non-dominated solutions. As shown in the figure, both approaches achieve seemingly similar approximation sets.

To deepen our analysis, we examine the impact of gradually increasing the percent-

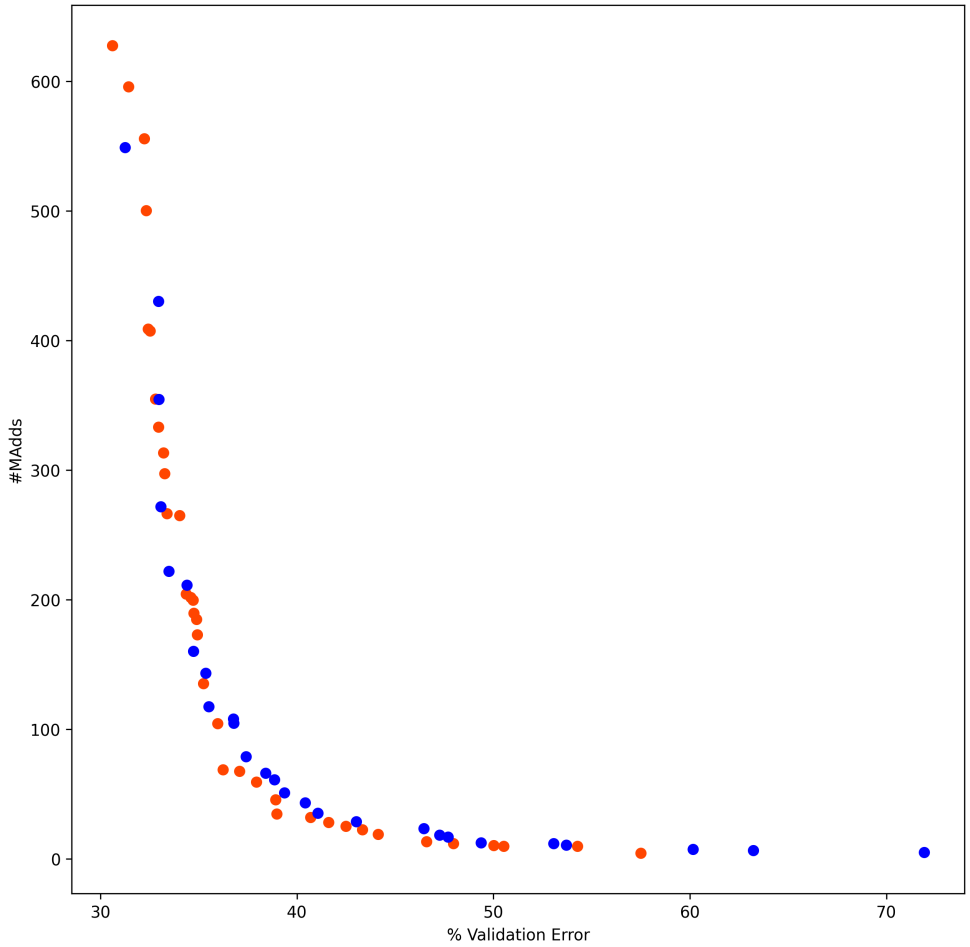


Figure 6.1: Illustration of the (aggregated) Pareto fronts obtained respectively by the baseline CGP-NASV2 (Blue) and our proposal (Orange).

age of data used in each generation on the multi-objective optimization process. We calculate the hypervolume (HV) [93] achieved by each method for each given generation of the NSGA-II algorithm. Specifically, we compute the Hypervolume Relative Deviation (HVRD) defined as $hvrd\left(A\right) = \left(hv\left(R\right) - hv\left(A\right)\right)/hv\left(R\right)$, where A represents the considered approximation set at a given generation and R is the best (reference) Pareto front approximation, derived by aggregating results across all executions. The reference point for the hypervolume is the Nadir point, obtained from the worst values for each objective. The lower the HVRD value, the better the approximation set and, consequently, the underlying approach.

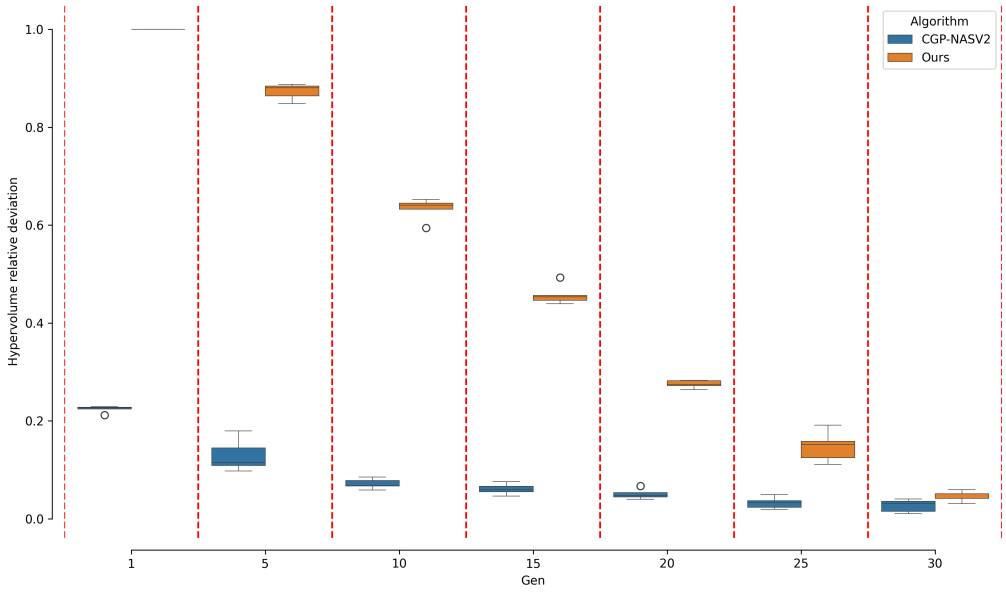


Figure 6.2: Box plot of the hypervolume relative deviation; in blue is the baseline CGP-NASV2, and in orange is our proposal across generations.

As reported in Figure 6.2, the HVRD of the CGP-NASV2 baseline is shown in blue, while our approach is represented in orange. It is important to interpret the difference in relative HVRD values from Figure 6.2 with caution, as the two approaches use different amounts of training data in the early generations but the same size of test data. Therefore, it might be unfair to conclude that one approach is dominating the other without retraining both approaches using the exact same amount of training data.

With this consideration, we notice a notable disparity in the distribution of solutions between our proposal and the baseline, primarily due to the limited amount of data used to train solutions in the early generations. However, using less data reduces the evaluation time for these solutions. As more data is utilized, solutions progressively converge towards the baseline. In the final generation, both approaches operate on an equal amount of data, resulting in minimal differences between solutions.

Thus, we conclude that this novel performance estimation strategy can effectively guide solutions throughout the evolutionary process. Even with limited training data, the proposed approach retains the ability to differentiate between good and bad solutions, evidenced by the continuous improvement of solutions over generations, eventually catching up to the more costly baseline approach.

Besides, such an observation applies not only to the Pareto set approximation but also to specific solutions extracted at each generation. In fact, Figure 6.3 illustrates the quality of the Best and Knee solutions obtained across generations for the two competing multi-objective NAS approaches. We observe that, in the case of our proposal, the classification error in early generations is higher compared to the baseline. This is again due to the smaller amount of data used to train these solutions. Nevertheless, we can still see clear changes between generations, unlike the baseline CGP-NASV2, where a faster convergence is evident and the solutions do not vary much; by generation 30, both algorithms show convergence. In the case of the Knee solutions, our proposal finds a better solution with respect to the classification error. This can, for instance, be seen in the bottom sub-figure of Figure 6.3.

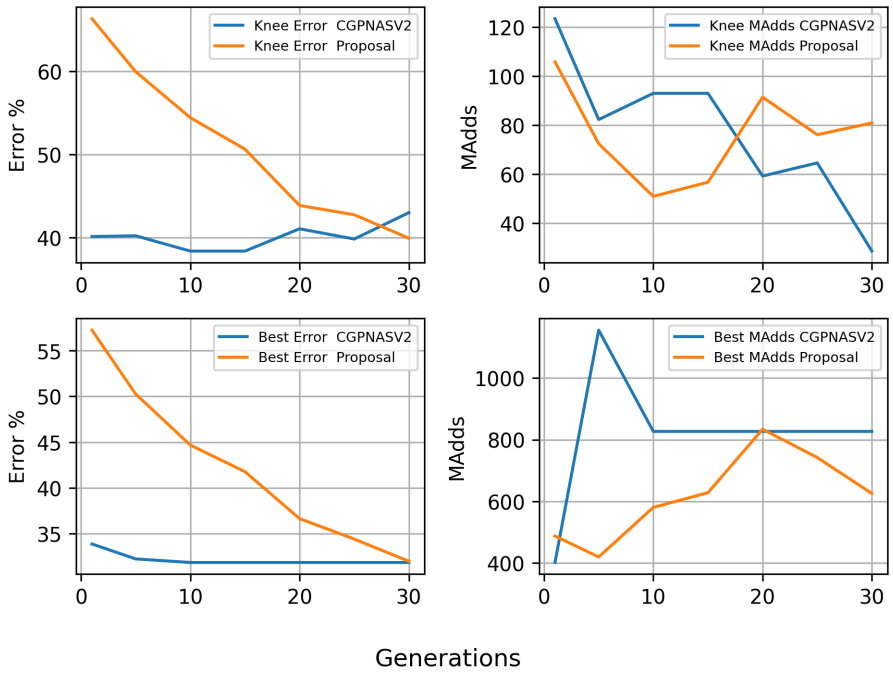


Figure 6.3: Selected solutions in each generation using the proposed estimation performance strategy. Both objectives are shown, as well as two types of solutions: knee solutions and best solutions. Top sub-Figure: our proposed method versus CGP-NASV2 in Knee solutions. Bottom sub-Figure: our proposed method versus CGP-NASV2 in Best solutions.

6.4 Discussion

In this Chapter, we proposed a straightforward strategy to estimate the performance of CNN architectures within the context of Neural Architecture Search (NAS). The proposed approach involves gradually increasing the size of the training dataset, thereby reducing the computational time required to train and evaluate the solutions obtained through the evolutionary process. Using the CGP-NASV2 representation as a baseline, we conducted a comprehensive comparison and observed that this new evaluation approach indeed accelerates the search process. By utilizing less data in the initial stages, the search is expedited. While this introduces some bias, which affects the quality of the solutions, the multi-objective NSGA-II algorithm is still capable of distinguishing between these solutions and searching for a relatively high-quality approximation set.

Chapter 7

Conclusions and Future work

In this thesis, we tackled the key components of Neural Architecture Search through multi-objective optimization, focusing on the development of novel approaches that address both performance and computational efficiency. The primary contribution of this work lies in the introduction of new representations for NAS, the design of an effective search space, and the implementation of a novel search and evaluation strategies. The following are the conclusions of this doctoral thesis work:

- **Novel search space for multi-objective NAS:**
  We introduced two new representations, CGP-NASV1 and CGP-NASV2, designed to offer greater flexibility and scalability in the NAS process. These representations were specifically tailored to navigate complex search spaces in a continuous domain, enabling the exploration of a diverse set of architectures, it was also tested with different MOEAs and it was observed that our representation is easily adaptable to different search algorithms. From the results obtained, in the three benchmarks tested we can observe that in general our proposal has a quite favorable performance, if we focus on the methodologies that use CGP as search space, our proposal is better, also, through the analysis of the Hypervolume we see a clear convergence of the solutions through the generations, finally we can conclude that indeed this new representation that includes hyperparameters, is adaptable and flexible which allows a better performance in NAS.

- **Progressive Self-Supervised multi-objective NAS**

  The proposed approach integrated two powerful strategies to enhance the efficiency and generalization capabilities of neural architecture search. First, leveraging *self-supervised learning* enabled the generation of more generalized architectures with the ability to transfer knowledge across different tasks. This significantly improved the versatility of the architectures, allowing them to perform well in a variety of scenarios and domains.

  Additionally, the introduction of a progressive NAS mechanism focused the search power and evolutionary operations on specific, promising regions of the architecture. This targeted exploration ensured a more efficient and focused search process, leading to the discovery of architectures that not only performed better but also met multiple performance criteria. Given the results obtained, we conclude that the architectures discovered by this method prioritize operations conducive to enhanced generalization, as evidenced by the Grad-CAM analysis. Also our method shows a significant reduction in GPU days, reaffirming also that the self-supervised learning strategy can reduce the overall search time.

- **A new performance estimation strategy:**
  Recognizing the high computational cost associated with NAS, we explored a novel performance estimation strategies aimed at reducing evaluation time. The strategy focused on *incremental training data usage* during the search process, progressively increasing the amount of data used for evaluation as the search progressed. This method significantly reduced computational time while maintaining the accuracy of the search, addressing a major bottleneck in NAS. We found that, by using this strategy, the MOEA can discern between solutions even if they have a high level of bias, which comes from the less information used to train the solutions in the early generations of the search. However, by linearly increasing the percentage of data, the search can refine the solutions, thus providing a set of highly competitive solutions, this is confirmed when we compared the total GPU-days with the baseline obtaining a speed up of 4.6.

7.1 Future Work

For future research, several research lines can be pursued to extend and improve the current work:

- **Comparison of evolutionary multi-objective algorithms:** We plan to compare different evolutionary multi-objective algorithms and explore hybridization with other methods, such as early stopping mechanisms, to further accelerate the search process. This aims to reduce computational time while maintaining the effectiveness of the search.
- **Dynamic adjustment of hyper-parameters:** Investigating dynamic adjustment of critical hyper-parameters in a more efficient and effective way. It is hoped that by performing the search these hyper-parameters can lead to better and less complex solutions., addressing one of the significant challenges in NAS design.
- **Incorporation of additional objectives:** Another direction is to enhance the search process by incorporating additional objectives, such as latency and model parameters. This will help better capture the trade-offs between various performance metrics, resulting in more balanced and comprehensive neural architectures.
- **Integration of local search algorithms:** We aim to integrate local search algorithms that can learn the weights of architectures, refining and improving the solutions generated by our NAS approach. This hybrid strategy could further optimize the architectures for various tasks.
- **Application of CGP-NASV2 to real-world challenges:** Finally, applying the CGP-NASV2 representation to real-world challenges, particularly in environments where architecture complexity is critical (e.g., mobile or embedded devices), is a priority. We aim to tailor our approach for resource-constrained scenarios, optimizing architectures for efficiency without sacrificing performance, thus broadening the applicability of our research.

Bibliography

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," Tech. Rep., 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [review article]," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[3] M. Kolbk, Z.-H. Tan, J. Jensen, M. Kolbk, Z.-H. Tan, and J. Jensen, "Speech intelligibility potential of general and specialized deep neural network based speech enhancement systems," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 25, no. 1, p. 153–167, jan 2017. [Online]. Available: https://doi.org/10.1109/TASLP.2016.2628641

[4] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[5] A. D. Martinez, J. Del Ser, E. Villar-Rodriguez, E. Osaba, J. Poyatos, S. Tabik, D. Molina, and F. Herrera, "Lights and shadows in Evolutionary Deep Learning: Taxonomy, critical methodological analysis, cases of study, learned lessons, recommendations and challenges," *Information Fusion*, vol. 67, pp. 161–194, mar 2021.

[6] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving Deep Neural Networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780128154809000153

[7] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, sep 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9201169/

[8] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-662-44874-8

[9] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net," in *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: ACM, jul 2019, pp. 419–427. [Online]. Available: https://dl.acm.org/doi/10.1145/3321707.3321729

[10] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search," in *LNCS*, vol. 12346 LNCS, aug 2020, pp. 35–51. [Online]. Available: https://doi.org/10.1007/978-3-030-58452-8

[11] C. Garcia-Garcia, H. J. Escalante, and A. Morales-Reyes, "CGP-NAS," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, vol. 1. New York, NY, USA: ACM, jul 2022, pp. 643–646. [Online]. Available: https://doi.org/10.1145/3520304https://dl.acm.org/doi/10.1145/3520304.3528963

[12] C. Termritthikun, Y. Jamtsho, J. Ieamsaard, P. Muneesawang, and I. Lee, "EEEA-Net: An Early Exit Evolutionary Neural Architecture Search," *Engineering Applications of Artificial Intelligence*, vol. 104, p. 104397, 2021. [Online]. Available: https://doi.org/10.1016/j.engappai.2021.104397

[13] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural AutoML for deep learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: ACM, jul 2019, pp. 401–409. [Online]. Available: https://dl.acm.org/doi/10.1145/3321707.3321721

[14] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," pp. 141–163, mar 2020.

[15] A. Bakhshi, S. Chalup, and N. Noman, *Fast Evolution of CNN Architecture for Image Classification*. Singapore: Springer Singapore, 2020, pp. 209–229. [Online]. Available: https://doi.org/10.1007/978-981-15-3685-4

[16] L. Xie and A. Yuille, "Genetic CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2017, pp. 1388–1397. [Online]. Available: http://ieeexplore.ieee.org/document/8237416/

[17] B. Wang, B. Xue, and M. Zhang, "Particle Swarm Optimization for Evolving Deep Convolutional Neural Networks for Image Classification: Single- and Multi-Objective Approaches," pp. 155–184, 2020.

[18] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[19] Y. Ho and D. L. Pepyne, "Simple explanation of the no free lunch theorem of optimization," *Kibernetika i Sistemnyj Analiz*, vol. 38, no. 2, pp. 164–173, 2002. [Online]. Available: https://link.springer.com/article/10.1023/A:1016355715164

[20] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A Survey on Evolutionary Neural Architecture Search," aug 2020. [Online]. Available: https://arxiv.org/abs/1611.01578http://arxiv.org/abs/2008.10937

[21] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions," *arXiv*, vol. 37, no. 111, p. 33, jun 2020. [Online]. Available: http://arxiv.org/abs/2006.02903

[22] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[23] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, jun 2002. [Online]. Available: http://mitpress.mit.edu/e-mailhttps://www.mitpressjournals.org/doi/abs/10.1162/106365602320169811

[24] D. Fogel, "What is evolutionary computation?" *IEEE Spectrum*, vol. 37, no. 2, pp. 26–32, feb 2000. [Online]. Available: https://ieeexplore.ieee.org/document/819926/

[25] H. Iba, *Evolutionary Computation and Meta-heuristics*. Singapore: Springer Singapore, 2020, pp. 3–33. [Online]. Available: https://doi.org/10.1007/978-981-15-3685-4_1

[26] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[27] J. Miller, P. Thomson, T. Fogarty, and I. Ntroduction, "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study," *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, 10 1999.

[28] J. F. Miller, "Cartesian Genetic Programming," in *Natural Computing Series*, 2011, vol. 43, pp. 17–34. [Online]. Available: http://link.springer.com/10.1007/978-3-642-17310

[29] C. Garcia-Garcia, A. Morales-Reyes, and H. J. Escalante, "Continuous cartesian genetic programming based representation for multi-objective neural architecture search," *Applied Soft Computing*, vol. 147, p. 110788, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623008062

[30] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

[31] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, ser. Genetic and Evolutionary Computation Series. Boston, MA: Springer US, 2007. [Online]. Available: http://link.springer.com/10.1007/978-0-387-36797-2

[32] K. Deb, *Evolutionary and Swarm Intelligence Algorithms*, ser. Studies in Computational Intelligence, J. C. Bansal, P. K. Singh, and N. R. Pal,

Eds. Cham: Springer International Publishing, 2019, vol. 779. [Online]. Available: http://www.springer.com/series/7092http://link.springer.com/10.1007/978-3-319-91341-4{_}6http://link.springer.com/10.1007/978-3-319-91341-4

[33] Z. Fan, W. Li, X. Cai, H. Li, C. Wei, Q. Zhang, K. Deb, and E. Goodman, "Push and pull search for solving constrained multi-objective optimization problems," *Swarm and Evolutionary Computation*, vol. 44, pp. 665–679, feb 2019.

[34] M. T. M. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," *Natural Computing*, vol. 17, no. 3, pp. 585–609, sep 2018. [Online]. Available: http://link.springer.com/10.1007/s11047-018-9685-y

[35] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," Tech. Rep. 2, 2002.

[36] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," 1993.

[37] J. Knowles and D. Corne, "The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 1, 1999, pp. 98–105 Vol. 1.

[38] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 3, no. 4, pp. 257–271, 1999.

[39] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," Tech. Rep., 2001.

[40] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, pp. 221–248, 1994.

[41] K. Deb, "Multi-objective Optimization," in *Search Methodologies*. Boston, MA: Springer US, 2014, pp. 403–449. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-6940-7{_}15

[42] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, dec 2007.

[43] Hui Li and Qingfu Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 284–302, apr 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4633340/

[44] N. Beume, B. Naujoks, and M. Emmerich, "Sms-emoa: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.

[45] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms — a comparative case study," in *Parallel Problem Solving from Nature — PPSN V*, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 292–301.

[46] J. G. Falcón-Cardona and C. A. Coello, "Indicator-based Multi-objective Evolutionary Algorithms: A Comprehensive Survey," jun 2020.

[47] N. Noman, *A Shallow Introduction to Deep Neural Networks*. Singapore: Springer Singapore, 2020, pp. 35–63. [Online]. Available: https://doi.org/10.1007/978-981-15-3685-4_2

[48] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[49] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[50] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis and Machine Vision*. Boston, MA: Springer US, 1993.

[51] S. Russell and P. Norvig, *Artifical Intelligence: A Modern Approach (Third Edition)*, 3rd ed., 2009.

[52] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, vol. 43, no. 11, pp. 4037–4058, nov 2021.

[53] J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun, H. Luo, and D. Tao, "A survey on self-supervised learning: Algorithms, applications, and future trends," 2023.

[54] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2536–2544.

[55] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 69–84.

[56] Z. Feng, C. Xu, and D. Tao, "Self-supervised representation learning by rotation feature decoupling," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 356–10 366.

[57] D. B. D'ambrosio and K. O. Stanley, *A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry*, 2007.

[58] S. Risi, J. Lehman, and K. O. Stanley, *Evolving the Placement and Density of Neurons in the HyperNEAT Substrate*, 2010.

[59] A. Torabi, A. Sharifi, and M. Teshnehlab, "Using Cartesian Genetic Programming Approach with New Crossover Technique to Design Convolutional Neural Networks," *Neural Processing Letters*, 2022. [Online]. Available: https://doi.org/10.1007/s11063-022-11093-0

[60] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely Automated CNN Architecture Design Based on Blocks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1242–1254, apr 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8742788/

[61] R. H. R. Lima, D. Magalhães, A. Pozo, A. Mendiburu, and R. Santana, "A grammar-based GP approach applied to the design of deep neural networks,"

Genetic Programming and Evolvable Machines, vol. 23, no. 3, pp. 427–452, sep 2022. [Online]. Available: https://doi.org/10.1007/s10710-022-09432-0https://link.springer.com/10.1007/s10710-022-09432-0

[62] Y.-H. Kim, B. Reddy, and S. Yun, "NEMO: Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy Chanwon Seo," Tech. Rep., 2017.

[63] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "Boa: The bayesian optimization algorithm." Morgan Kaufmann, 1999, pp. 525–532.

[64] S. Li, Y. Sun, G. G. Yen, and M. Zhang, "Automatic Design of Convolutional Neural Network Architectures Under Resource Constraints," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9609007/

[65] Y. Xue, P. Jiang, F. Neri, and J. Liang, "A Multi-objective evolutionary approach based on graph-in-graph for neural architecture search of convolutional neural networks," *International Journal of Neural Systems*, vol. 31, no. 9, sep 2021.

[66] Q. Liu, X. Wang, Y. Wang, and X. Song, "Evolutionary convolutional neural network for image classification based on multi-objective genetic programming with leader–follower mechanism," *Complex and Intelligent Systems*, 2022.

[67] M. Pinos, V. Mrazek, and L. Sekanina, "Evolutionary approximation and neural architecture search," *Genetic Programming and Evolvable Machines*, jun 2022. [Online]. Available: https://doi.org/10.1007/s10710-022-09441-zhttps://link.springer.com/10.1007/s10710-022-09441-z

[68] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: ACM, jul 2017, pp. 497–504. [Online]. Available: https://dl.acm.org/doi/10.1145/3071178.3071229

[69] W. Irwin-Harris, Y. Sun, B. Xue, and M. Zhang, "A Graph-Based Encoding for Evolutionary Convolutional Neural Network Architecture Design," in *2019 IEEE*

Congress on Evolutionary Computation (CEC). IEEE, jun 2019, pp. 546–553. [Online]. Available: https://ieeexplore.ieee.org/document/8790093/

[70] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural Architecture Transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9328602/

[71] J. Clegg, J. A. Walker, and J. F. Miller, "A new crossover technique for Cartesian genetic programming," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*. New York, New York, USA: ACM Press, 2007, p. 1580. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1276958.1277276

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[73] M. Tan and Q. V. Le, "Efficientnetv2: Smaller models and faster training," 2021. [Online]. Available: https://arxiv.org/abs/2104.00298

[74] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2019. [Online]. Available: https://arxiv.org/abs/1709.01507

[75] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019. [Online]. Available: https://arxiv.org/abs/1801.04381

[76] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[77] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017. [Online]. Available: https://arxiv.org/abs/1610.02357

[78] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *International Conference on Learning Representations (ICLR)*, 2016.

[79] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[80] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[81] C. Garcia-Garcia, M.-G. Martínez-Peñaloza, and A. Morales-Reyes, "cMOGA/D: a novel cellular GA based on decomposition to tackle constrained multiobjective problems." in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: ACM, jul 2020, pp. 1721–1729. [Online]. Available: https://doi.org/10.1145/3377929.3398137

[82] F. E. Fernandes Jr. and G. G. Yen, "Pruning Deep Convolutional Neural Networks Architectures with Evolution Strategy," *Information Sciences*, vol. 552, pp. 29–47, apr 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025520310951

[83] W. Haynes, *Bonferroni Correction*. New York, NY: Springer New York, 2013, pp. 154–154. [Online]. Available: https://doi.org/10.1007/978-1-4419-9863-7_1213

[84] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[85] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[86] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," mar 2017. [Online]. Available: http://arxiv.org/abs/1703.01041

[87] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-Assisted Evolutionary Deep Learning Using an End-to-End Random Forest-Based Performance Predictor," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 350–364, apr 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8744404/

[88] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *CoRR*, vol. abs/1605.07648, 2016. [Online]. Available: http://arxiv.org/abs/1605.07648

[89] S. Zagoruyko and N. Komodakis, "Wide residual networks," *CoRR*, vol. abs/1605.07146, 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[90] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," 2018. [Online]. Available: https://arxiv.org/abs/1810.03505

[91] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626.

[92] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "CINIC-10 is not imagenet or CIFAR-10," *CoRR*, vol. abs/1810.03505, 2018. [Online]. Available: http://arxiv.org/abs/1810.03505

[93] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *Evolutionary Computation, IEEE Transactions on*, vol. 7, pp. 117 – 132, 05 2003.