POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Zero-Cost Proxies in Neural Architecture Search: A Comprehensive Study and Design of a novel hybrid proxy

Supervisors

Prof. Lia MORRA

Vassili MAILLET

Marc RAVAINE

Johann BARBIER

Candidate

Eleonora POETA

Academic Year 2022-2023

Ai miei nonni Fabrizio e Bruna,
esempio di saggezza e amore.

Abstract

Automated machine learning (AutoML) is a new, rapidly growing area of machine learning (ML) that aims to automate the creation of machine learning pipelines, including the design, training and deployment of machine learning models, model selection and hyperparameter tuning. Neural Architecture Search (NAS) is a sub-field of AutoML that focuses specifically on automating the design of neural network architectures. NAS has become increasingly popular due to its potential to significantly improve the performance of deep learning models by discovering optimal neural network architectures. However, the search process can be computationally expensive as it requires training and evaluating a large number of architectures. This is where Zero-cost proxies come in. They are low-fidelity approximation techniques that can be used to estimate the performance of a candidate architecture without the need for full training. Zero-cost (ZC) proxies significantly lower the computational cost of the search procedure and can result in a more effective and efficient NAS.

In this thesis, AutoML and NAS are exhaustively presented providing the reader with the necessary context for understanding the Zero-cost proxies. Next, existing Zero-cost proxies are exposed and investigated, followed by the research and analysis of the newly proposed Hybrid ZC proxy. The Hybrid Zero-cost proxy is obtained by combining the benefits of both Data-Independent (DI) and Data-Dependent (DD) ZC proxies. Specifically, the Hybrid ZC proxy is created through a weighted sum of LogSynflow [1], which is the best performing ZC proxy DI, and Nwot [2] , which is the best performing ZC proxy DD.

To make a quantitative comparison with the other existing ZC proxy metrics, several tests are conducted to evaluate the performances on various reference datasets. Specifically, the performance of each ZC proxy are measured by calculating two correlation coefficients, namely Kendall-tau and Spearman, between the score obtained by the neural network architecture with the ZC proxy and the accuracy achieved by training and testing the model. The experimental results show that the proposed Hybrid Zero-cost proxy outperforms the LogSynflow and Nwot ZC proxies in terms of Kendall-tau and Spearman correlation values, achieving higher correlation values.

Overall, the Hybrid Zero-cost proxy proposed in this Master's thesis show highly promising performance and outperform its main competitors, LogSynflow and Nwot, in all three tested datasets. These results suggest that the proposed solution could be a crucial factor in enabling the development of faster and more effective Automated machine learning systems. Further research and experimentation are needed to confirm its potential impact, but these initial findings are a strong indication of its possibilities.

Acknowledgements

I want to start by expressing my gratitude to Prof.ssa Lia Morra, for her availability, assistance, and tutoring with this thesis.

Thank you also to the people at ALTEN Lab Sophia Antipolis for the opportunity to work with them and for sharing their resources with me, which made this work possible.

I would like to thank my family, this accomplishment would definitely not have been possible without your care and guidance, thank you for always being there to support me at every step of the way.

I am really grateful to my boyfriend Simone.Thank you for always supporting and standing by me with infinite patience and love.

I would like to say a big thank to my wonderful friends Arianna, Beatrice, Michela, Micaela and Alessia. Thanks for being with so long, I love you.

Then, I would like to thank all the friends that who walked with me through those university years. Sorry if I will just make a list, but you already know how much you mean to me, so thanks to: Nour, Richi, Fra, Fede, Marti, Ale, Pasto, AleSanna, Leo, AndreTorre, AndreLai, Paoletta.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This Master's thesis work was carried out at ALTEN Lab of Sophia Antipolis, France. There are four ALTEN Labs, located throughout France, that are committed to supporting cutting-edge research and technology initiatives as well as creative approaches to managing businesses' upcoming digital transformation. In this innovative context we find the System Behaviour Management (SBM) project, in which I was included. The SBM project was created by one of my supervisors, Marc RAVAINE, and it aims to automate the creation of a complete Machine Learning pipeline, for different types of problems. In particular, this project wants to facilitate the work made by data engineers, in which a lot of experience is needed, and, in the long run, to empower non-expert people to interface with the data science world. There are numerous tasks to be completed as part of this project, but the work of this Master's thesis is concentrated on Neural Architecture Search (NAS), particularly on the idea of speeding up this task.

Automated Machine Learning (AutoML) is a rapidly expanding field whose goal is to automate the process of developing machine learning models. Many of the time-consuming and laborious machine learning tasks, such as feature engineering, model selection, and hyperparameter tuning, are automated by AutoML methods. Hence, AutoML aims to make it easier for non-experts to create high-quality machine learning models by automating the most difficult parts of the process. AutoML is a broad field that encompasses a range of subfields, each focused on automating a different aspect of the machine learning process.

Neural Architecture Search (NAS) is a sub-field of AutoML that focuses specifically on automating the design of neural network architectures. In the traditional neural network design, the architecture is chosen manually by a human expert, based on their understanding of the problem and their intuition. It is possible to categorize different methods for NAS according to three dimensions:

1. Search Space: The search space is the collection of all neural network architectures that can be investigated during the NAS process. The level of detail in the architecture specification may range from high-level descriptions, such as cells or blocks, to detailed specifications of individual layers in this area.

2. Search Strategy: The search strategy is the process used to sift through the search space and find the ideal neural network architecture. This can involve methods like gradient-based optimization, Bayesian optimization, reinforcement learning, and evolutionary algorithms.

3. Performance Estimation: The techniques used to assess the candidate neural network architectures' performance during the NAS process, are known as performance estimation. This can be accomplished by training and analysing each architecture on a validation set, or by using proxies that allow fast evaluation .

The use of neural architecture search (NAS) has become increasingly popular in recent years. However, this process can be computationally expensive and time-consuming, making it difficult to apply to real-world problems. One solution to this problem is to use low-cost or zero-cost proxies for expensive performance evaluations. These proxies are designed to estimate the performance of a neural network architecture with minimal computational resources, allowing for a more efficient search process. In [3] Abdelfattah *et al.* proposed a set of Zero-cost proxies that can predict the model score with just a small batch of training data. The predicted score is correlated to the accuracy of the model on the test data. As a result, estimating the score ultimately involves estimating how well the model will perform given a specific set of data.

This Master's thesis aims to investigate and analyze the Zero-cost proxy metrics currently available in the field. Additionally, this thesis proposes a novel Hybrid Zero-cost proxy metric that leverages the benefits of both data-independent and data-dependent categories of ZC proxies.

The structure of the thesis will be the following:

- Chapter 1: Introduction to the thesis.

- Chapter 2: Introduction to Machine Learning and Deep Learning.

- Chapter 3: Problem Definition that contains an overview of AutoML, NAS and the Zero-Cost proxies.

- Chapter 4: Related Works about Zero-cost proxies metrics that were researched while developing the thesis.

- Chapter 5: Presentation of the methodology used to create the Hybrid Zero-cost proxy, as well as the experiments that were conducted to evaluate its performance.
- Chapter 6: Display of the results achieved and comparison to other state-of-the-art.
- Chapter 7: A brief discussion of the results and possible future works are proposed.

Chapter 2

Introduction to Machine Learning and Deep Learning

2.1 Introduction to Machine Learning

Machine learning (ML) is a sub-field of Artificial Intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to learn from and make predictions or decisions based on data, without being explicitly programmed to do so.

Tom Mitchell, a renowned researcher in the field of machine learning, provided a definition of machine learning that is generally accepted as a standard in the field. In his book "What is Machine Learning?" he defines machine learning as

> “A computer program is said to learn from experience E with respect to a class of tasks T and a performance measure P if its performance at tasks in T, as measured by P, improves with experience E.”

In simpler terms, Mitchell’s definition states that machine learning occurs when a computer program improves at a task through experience and by using a performance measure. This definition highlights the key characteristics of machine learning, which include the use of experience, tasks, and performance measures to drive improvement.

2.1.1 Machine Learning approaches

According to Tom Mitchell’s definition of machine learning, it is possible to distinguish three main approaches.

- **Supervised Learning**: In supervised learning approach, the computer is instructed by example. The algorithm is trained on labeled data, where the correct output is already known. The goal of the algorithm is to learn the mapping between input and output, and then use this mapping to make predictions on new, unseen data.

- **Unsupervised Learning**: In this approach, there are no labelled data providing instruction to the algorithms. Indeed, the algorithm is trained on unlabeled data and the goal is to identify patterns and relationships within the data. The algorithm is not given any specific output to learn, but instead must identify structure and meaningful relationships in the data on its own.
- **Semi-supervised Learning**: Semi-supervised learning is another approach in machine learning that is a hybrid of supervised and unsupervised learning. In semi-supervised learning, the algorithm is trained on a combination of labeled and unlabeled data. The labeled data provides the algorithm with a basic understanding of the mapping between input and output, while the unlabeled data provides additional information that can be used to refine the algorithm's understanding.
- **Reinforcement Learning**: In this approach it is exploited a learning process in which the algorithm is trained by making decisions and receiving feedback in the form of rewards or penalties. The algorithm learns through trial-and-error and gradually improves its decision-making ability over time.

2.1.2 Machine Learning tasks

There are different tasks in machine learning, and the choice of the task depends on the type of data, the goals of the project, and the available resources. Machine learning algorithms can also be combined to perform more complex tasks. The following are the main tasks for which Machine learning is used to solve.

- **Classification**: Classification task consists in categorizing data into one of several predefined classes. The algorithm would use its training data to learn the features that distinguish each class, and then apply this knowledge to new images to make predictions. This task is commonly used in a variety of real-world applications such as image classification, spam detection, and credit card fraud detection. Classification is a widely used task in machine learning, and there are many algorithms and techniques that can be used to perform this task, including decision trees, random forests, support vector machines, and neural networks. The choice of algorithm depends on the type of data and the requirements of the specific task.
- **Regression**: Regression is a task in machine learning that is used to predict a continuous output value for a given input. Its goal is to model the relationship

between the input variables and the output variable, and then use this model to make predictions on new data. Regression is widely used in a variety of real-world applications such as stock price prediction, weather forecasting, and house price prediction. For example, the possible features to predict the price of a house can be the size, location, age, and other features. There are many algorithms that can be used for regression, including linear regression, polynomial regression, decision trees, random forests, and neural networks.

- **Clustering**: Clustering is a task in machine learning that is used to group data points with similar characteristics into clusters. The goal of clustering is to identify patterns and structure in the data, and to divide the data into groups or clusters based on similarity. The algorithm does not have any prior knowledge of the number of clusters, or what each cluster represents. Instead, it must learn the structure of the data and determine the number and composition of the clusters on its own. Clustering can be used in customer segmentation, where a company wants to divide its customers into different groups based on their behavior and demographics. The algorithm would analyze the data on the customers, such as their age, income, location, and spending habits, and then divide them into groups that are similar to each other. There are many algorithms that can be used for clustering, including k-means and hierarchical clustering.

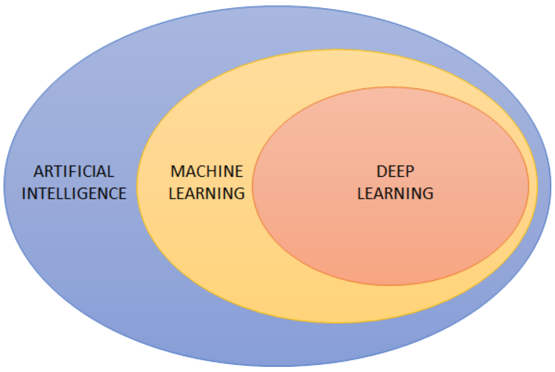

Figure 2.1: The interplay between Artificial Intelligence, Machine Learning and Deep Learning.

2.2 Introduction to Deep Learning

Deep learning (DL) is a relatively new sub-field of machine learning that has been growing in popularity in recent years. Deep learning has achieved remarkable success in a variety of applications, including computer vision, natural language processing, and speech recognition. One of the key advantages of deep learning is its ability to learn high-level abstractions from large amounts of data, making it well-suited to tasks that require understanding and processing complex and unstructured data, such as images, videos, and audio signals. Deep learning uses Artificial Neural Networks (NN) with multiple layers, hence "deep", to learn hierarchical representations of data. Each layer of deep learning models is responsible for learning a different level of abstraction. The lower layers learn basic features such as, in an image, edges and shapes, while, the higher layers, learn higher-level features such as objects and semantic concepts. This hierarchical representation of data enables deep learning models to perform complex tasks with high accuracy.

2.3 From Artificial Neural Networks to Deep Learning models

Deep Learning models and Artificial Neural Networks (ANNs) are two categories of machine learning models that draw their inspiration from the structure and operation of the human brain. ANNs are made up of interconnected artificial neurons that are used to learn relationships between inputs and outputs.

Deep learning models are a type of ANN that consists of multiple layers of artificial neurons that allow them to learn hierarchical data representations and increasingly complex features as the data flows through the network. Deep learning models have several advantages over traditional artificial neural networks. For example, the usage of non-linear activation functions, such as the rectified linear unit (ReLU), allows the model to learn non-linear relationships between inputs and outputs. This makes deep learning models well-suited to tasks where the relationship between inputs and outputs is non-linear. Additionally, deep learning models are capable of learning from large amounts of data, making them well-suited to tasks such as image and speech recognition, natural language processing, and recommendation systems. Overall, the advantages of deep learning models have made them a critical component in many artificial intelligence systems and have driven the recent advancements in artificial intelligence. The following section delves into one of the cornerstone models of deep learning, the Convolutional Neural Networks

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), introduced by LeCun in 1989, are a type of deep learning model that excel in computer vision and natural language processing tasks. The name "convolutional" refers to the use of mathematical convolution operations, rather than simple matrix multiplication, to analyze and understand the input data. In computer vision, CNNs can take an image as input and use a series of layers to identify and classify different objects or aspects within the image. The network learns to assign importance, through weights and biases, to different regions of the image, allowing it to differentiate between objects and background. There exist different types of layers:

- **Convolutional Layers**: as stated previously, MLPs usually have fully connected layers, where every neuron takes as input every output from the previous layer, and this implies a great amount of parameters for the deep networks. On the contrary, there are the CNNs that have Convolutional layers, where every neuron is in charge of a specific sub-window of data which "stride" through the set. The objective of the convolution operation is to extract the high-level features such as edges, from the input image. CNNs need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the low-level features such as edges, color, gradient orientation, etc. As more layers are added, the architecture captures to the higher-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

- **Pooling Layers** : after generating the convolutional output, a pooling layer may be inserted. The main purpose of the pooling layers is to progressively reduce the spatial size of the input image, so that number of computations in the network are reduced. This layer has the task of dividing the output in other sub-windows and then collapse each of them in a single data point with a specific function, that can usually be the average, sum or max.

- **Normalization Layers**: normalization layers in the CNNs often helps to speed up and stabilize the learning process. There are different types of normalization layers such as Batch Normalization (BN), Weight Normalization (WN), Weight Standardization(WS). The Batch Normalization (BN) is the most widely used and is responsible for transforming each input in the current mini-batch of data, by subtracting the input mean in the current mini-batch and dividing it by the standard deviation.

- **Non-linear activation Functions** : the purpose of the non-linear activation functions is to introduce non-linearity into the network. ReLU is increasingly the default choice of activation functions.

Chapter 3

Problem Definition

3.1 Introduction to Automated Machine Learning

Automated Machine Learning (AutoML) is a rapidly growing machine learning field that aims to make the process of designing, training, and deploying machine learning models more accessible and efficient. With the increasing demand for machine learning in various industries, the need for automating the creation of models has become increasingly important. An informal definition of an AutoML system is given by Prof. Frank Hutter *et al.*[4]:

Definition 3.1.1 (AutoML system) *Given:*

- *A Dataset*
- *A Task (e.g., regression or classification)*
- *A Cost Metric (e.g., accuracy or RMSE)*

an AutoML system automatically determines the approach that performs best for this particular application.

An AutoML system is a really valuable tool for organizations that want to leverage the power of machine learning but do not have the specialized knowledge or expertise in-house. With AutoML, they can build and deploy machine learning models more quickly and easily, allowing them to scale their machine learning efforts more effectively. Different advantages can be identified, including:

- Simpleness of use : AutoML offers a user-friendly interface that walks users through the process of creating, training, and deploying machine learning models, so users no longer need to possess specialised knowledge or expertise in the field to do so.

- Enhancement of performance and efficiency : By automating many of the manual and time-consuming tasks involved in building machine learning models, AutoML can speed up the model development process.
- Improved Reproducibility : By automating many of the manual and time-consuming tasks involved in building machine learning models, AutoML can speed up the model development process and help organizations scale their machine learning efforts more quickly.
- Reduced Risk of Human Error : The automation of difficult and complex tasks involved in machine learning model's creation can help reduce the risk of human error, resulting in more reliable and accurate results.
- Better Utilization of Expertise : AutoML allows data scientists and engineers to focus on more trivial tasks, such as feature engineering and model interpretation, instead of spending time on manual, repetitive tasks.

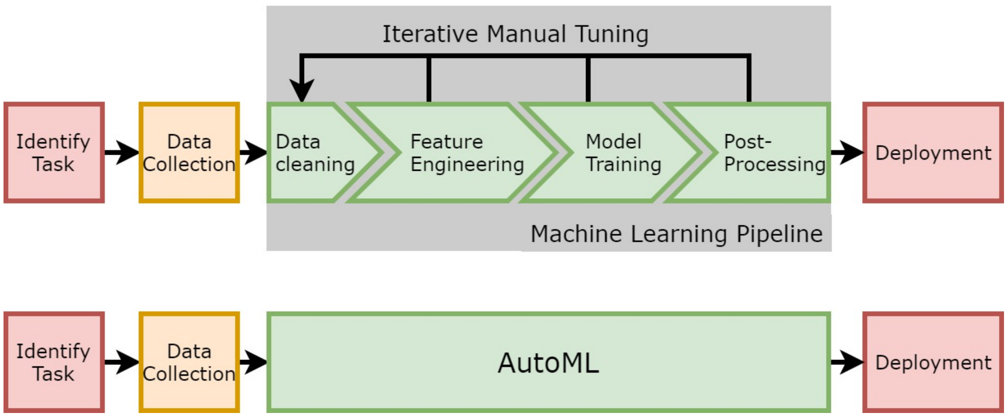

Figure 3.1: Macro distinctions between an AutoML pipeline and a ML pipeline. Source:[4]

Figure 3.1 depicts the distinctions between a ML pipeline (above) and an AutoML pipeline. In order to achieve state-of-the-art performance, the iterative manual tuning of green tasks within the machine learning pipeline must be repeated for each new dataset, which is very time-consuming. With AutoML, the time-consuming aspect is diminished because green tasks are automated and the effectiveness of developing new ML applications is improved, possibly even achieving better performance. In summary, AutoML is useful because it makes the process of building machine learning models more accessible, efficient, and accurate.

There exists risks, even when using AutoML, and it is important to be aware of them. As previously said, one of the main goals of AutoML is to make it simple for people who are not machine learning experts. However, achieving this goal could have a number of drawbacks, such as the application of AutoML without doing any research, falling in the production of models that are unreliable, unfair, or biased. It is therefore essential know how to manage this powerful tool keeping from having too much faith in it.

In terms of possible real-world applications of AutoML, we can mention the health sector, specifically image diagnosis using AutoML. Currently, image diagnosis is performed using machine learning involving the manual process of selecting and annotating a dataset of images, preprocessing and transforming the images, selecting an appropriate model architecture, and training the model on the preprocessed data. The model is then tested and evaluated, and the process is repeated until satisfactory results are achieved. This process can be really time-consuming and requires a high level of expertise in both machine learning and medical imaging. With AutoML, the entire process can be simplified by automating diagnosis based on medical images or clinical data from the patient, as well as speeding up the patient's subsequent treatment plan.

3.2 AutoML sub-fields

AutoML is made up of sub-fields, each of which is involved in a task of the overall system. The goal of an AutoML system is to optimise a cost function that accounts for all costs associated with the various sub-fields. An AutoML system include the following tasks that have to be investigated:

- Hyperparameter Optimization (HPO).
- Model and algorithm selection.
- Dynamic approaches (DAC).
- Neural Architecture Search (NAS).

The following paragraphs briefly describe Hyperparameter Optimization, Model and Algorithm selection and Dynamic approaches, and then space is given to NAS, the central topic of study in this thesis work.

3.2.1 Hyperparameter Optimization - HPO

Hyperparameter Optimization is a sub-field of AutoML that deals with the automated tuning of the hyperparameters of machine learning models. Every machine learning system has hyperparameters and the most basic task of AutoML is to automatically set these hyperparameters that have a crucial impact on how well the model performs. A formal definition of HPO can be:

Definition 3.2.1 (Hyperparameter Optimization [4]) *Let:*

- *λ be the hyperparameters of an ML algorithm A with domain Λ,*
- *D_{opt} be a dataset which is split into D_{train} and D_{val},*
- *$c(A_\lambda, D_{train}, D_{val})$ denote the cost of A_λ trained on D_{train} and evaluated on D_{val}.*
  The Hyperparameter optimization (HPO) problem is to find a hyperparameter configuration that minimizes this cost:

$$\lambda^* \in argmin_{\lambda \in \Lambda}\ c\left(A_\lambda, D_{train}, D_{valid}\right)$$

Due to the expanding use of machine learning in companies, HPO is also of significant commercial interest and assumes an increasingly significant role, whether in internal company tools, as a component of machine learning cloud services, or as a service in and of itself.

HPO faces several challenges, making it a difficult problem to solve in practise. For example the function evaluations can be extremely expensive for large models, complex machine learning pipelines, or large datasets. Another issue could be the configuration space of hyperparameters. This is frequently represented as a combination of continuous, categorical, and conditional hyperparameters, resulting in a really large space, difficult to deal with. Some commonly used HPO algorithms include Grid Search, Random Search, Bayesan Optimization, Evolutionary algorithms and so on.

3.2.2 Model and algorithm selection

The model and algorithm selection in AutoML is a crucial part of the overall AutoML process. The goal is to find the best model that performs optimally on the given data and provides accurate results for the problem at hand. The selection of algorithm and hyperparameters are obviously linked, and, in this matter, it is important to mention the existence of CASH.

CASH (Combined Algorithm Selection and Hyperparameter Optimization) is a type of AutoML approach that combines the tasks of algorithm selection and hyperparameter optimization. We can provide a definition below:

Definition 3.2.2 (CASH [4]) *Let:*

- $A = \{A_1, A_2, ..., A_k\}$ *be the set of algorithms,*
- Λ *be the set of hyperparameters of each machine learning algorithm* A_i*,*
- D_{opt} *be a dataset which is split into* D_{train} *and* D_{val}*,*
- $c(A_\lambda,\, D_{train},\, D_{val})$ *denote the cost of* A_λ *trained on* D_{train} *and evaluated on* D_{val}*.*
  we want to find the best combination of algorithm $A \in A$ *and its hyperparameter configuration* $\lambda \in \Lambda$ *minimizing:*

$$(A^*, \lambda^*) \in argmin_{A \in A, \lambda \in \Lambda}\ c\,(A_\lambda, D_{train}, D_{valid})$$

The goal of CASH is to find the best algorithm and set of hyperparameters for a given machine learning task in a single run, rather than performing algorithm selection and hyperparameter tuning as separate steps. This strategy can result in better outcomes, lower computational costs, and a better comprehension of the connections between algorithms, hyperparameters, and performance.

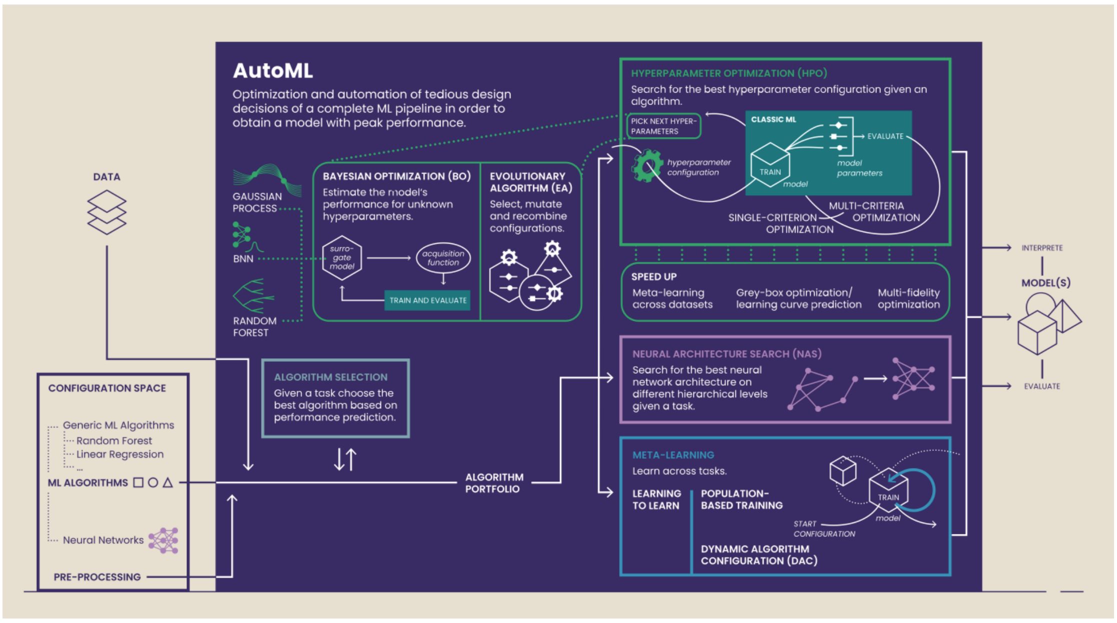
AutoML
Optimization and automation of tedious design decisions of a complete ML pipeline in order to obtain a model with peak performance.
DATA
GAUSSIAN PROCESS
BNN
RANDOM FOREST
BAYESIAN OPTIMIZATION (BO)
Estimate the model's performance for unknown hyperparameters.
surro-gate model
acquisition function
TRAIN AND EVALUATE
EVOLUTIONARY ALGORITHM (EA)
Select, mutate and recombine configurations.
HYPERPARAMETER OPTIMIZATION (HPO)
Search for the best hyperparameter configuration given an algorithm.
PICK NEXT HYPER-PARAMETERS
CLASSIC ML
EVALUATE
hyperparameter configuration
TRAIN
model
model parameters
MULTI-CRITERIA OPTIMIZATION
SINGLE-CRITERION OPTIMIZATION
SPEED UP
Meta-learning across datasets
Grey-box optimization/ learning curve prediction
Multi-fidelity optimization
INTERPRETE
MODEL(S)
EVALUATE
NEURAL ARCHITECTURE SEARCH (NAS)
Search for the best neural network architecture on different hierarchical levels given a task.
CONFIGURATION SPACE
Generic ML Algorithms
Random Forest
Linear Regression
ML ALGORITHMS
Neural Networks
PRE-PROCESSING
ALGORITHM SELECTION
Given a task choose the best algorithm based on performance prediction.
ALGORITHM PORTFOLIO
META-LEARNING
Learn across tasks.
LEARNING TO LEARN
POPULATION-BASED TRAINING
TRAIN
model
START CONFIGURATION
DYNAMIC ALGORITHM CONFIGURATION (DAC)

Figure 3.2: Description of the placement and communication of the different AutoML tasks. Source:[4]

HPO and CASH are two crucial components of AutoML that are essential to the effectiveness and performance of machine learning models. HPO is the process of enhancing a model's performance by optimising its hyperparameters. Additionally, CASH is a subset of AutoML that focuses on choosing the model architecture as well as the hyperparameters. Given the combination of strengths of these two approaches, AutoML provides a comprehensive solution for model selection and optimization, making it easier and faster to achieve high-quality machine learning models. Neural Architecture Search (NAS), represents the final step. It extends the idea of HPO and CASH by not only searching and tuning the hyperparameters of a predefined neural network architecture, but also searching for the optimal architecture itself. Before delving deeper into NAS, it is important to place all of the elements presented thus far within the AutoML pipeline. Figure 3.2, taken from the AutoML Lectures of F.Hutter *et al.* [4], shows, in detail, how the various tasks (HPO, NAS etc.) interact with each other and, more generally, where they are located within the AutoML pipeline. It is important to emphasise that there is no single way to build an AutoML pipeline and that the one depicted in the figure is intended to give the reader only an idea of what steps to follow.

3.3 Neural Architecture Search: definition and overview

Neural Architecture Search (NAS) is a sub-field of AutoML that is responsible of automating the design of artificial neural networks. It aims to find the optimal combination of different architectural elements, such as layer type, number of layers, number of neurons, and activation functions, in order to obtain a neural network that performs well on a particular dataset. NAS has become increasingly popular in recent years due to its ability to discover novel and effective architectures that outperform traditional hand-designed models, particularly for complex tasks such as computer vision and natural language processing. Using NAS can lead to better performance compared to manually selecting architectures, as well as saving time and resources compared to training and evaluating multiple architectures manually. A formal definition of NAS is given by the AutoML lectures of F. Hutter *et al.*[4]:

Definition 3.3.1 (NAS [4]) *Let:*

- *λ be an architecture for a deep neural network N with domain Λ,*
- *D_{opt} be a dataset which is split into D_{train} and D_{val},*
- *$c(N_\lambda, D_{train}, D_{val})$ denote the cost of N_λ trained on D_{train} and evaluated on D_{val}.*

The neural architecture search (NAS) problem is to find an architecture that minimizes this cost:

$$\lambda^* \in argmin_{\lambda \in \Lambda}\ c\left(N_\lambda, D_{train}, D_{valid}\right)$$

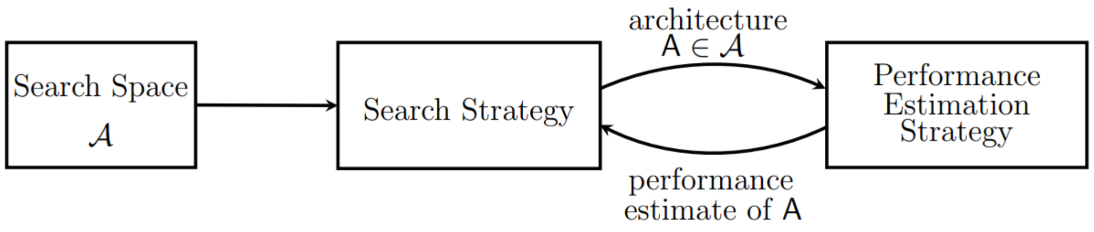

Figure 3.3: Illustration of Neural Architecture Search dimensions. Source:[5]

It is possible to categorize methods for NAS according to three dimensions: search space, search strategy, and performance estimation strategy. Figure 3.3 depicts the three NAS dimensions:
A Search Strategy selects an architecture from a predefined Search space A. Then, the architecture is passed through a Performance Estimation Strategy, which returns an estimate of the performance of the chosen architecture to the Search Strategy in order to improve the search policy adopted.

- Search Space: The search space defines which architectures are considered.
- Search Strategy: The search strategy details how to explore the search space.
- Performance Estimation Strategy: The objective of NAS is typically to find architectures that achieve high predictive performance on unseen data and performance estimation refers to the process of estimating this performance.

The following sections go into great detail on each of the three dimensions.

3.3.1 Search Space

NAS aims to search for a robust and well-performing neural architecture, by selecting and combining different basic elements from a predefined search space. Incorporating prior knowledge about properties well-suited for a task can reduce the size of the search space and simplify the search. This, however, also introduces a human bias, which might prevent the discovery of novel structural elements that go beyond the state of human knowledge.
The simplest search space is composed of chain-structured neural networks, illustrated in figure 3.4, on the left. A chain-structured neural network architecture

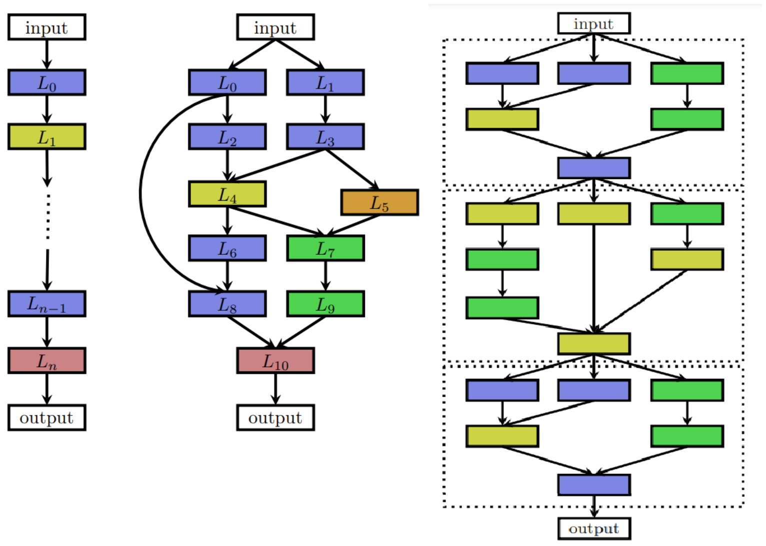

Figure 3.4: Illustration of an element of a search space of chain-structured neural network architectures (left) and, a more complex neural network architecture with multiple branches and skip connections (middle). On the right, a cell-based neural architecture. Source:[5]

A is composed of sequence of n layers, where the i_{th} layer, L_i, receives its input from layer *(i-1)*, L_{i-1} , and its output serves as the input for layer *(i+1)*, L_{i+1}. The search space is, therefore, parameterized by:

- The number of layers n.
- The type of operation each layer executes.
- The hyperparameters related to the operations.

The selection of operations and hyperparameters is closely linked, as the combination of both can lead to a search space that is not fixed length, but rather a conditional space. The search space can become more complicated with the addition of multiple-branches, i.e. having multiple parallel paths or branches within the network, each with its own set of operations and computations, and skip-connection operations, as shown in figure 3.4 in the middle. The skip-connections, introduced by He *et al.* in [6], allows the gradient flow from earlier layers to reach later layers without being altered by the activation functions in between. The idea behind skip-connections is to overcome the vanishing gradients problem in deep networks, which can make it difficult for the network to learn. In NAS, the presence of skip-connections results in architectures that can be theoretically infinitely complex, both in the number of layers, but also in the operations that consider layers even further back in the network.

Cell-based Search Space In this respect, the works [7] and [8] by Zoph *et al.* represents a turning point, with the introduction of the concept of the cell-based search space. In the cell-based search space, the neural network architecture is represented by a set of predefined building blocks called "cells". The final architecture can be created by combining and repeating these cells in different configurations. An example of a cell-based search space architecture is reported in figure 3.4, on the right. Comparatively to exhaustively combing through all conceivable combinations of operations and hyper-parameters, this enables a more flexible and scalable method of looking for the ideal architecture. The introduction of the cell-based search space was made in an effort to simplify the search space and improve the effectiveness and efficiency of the NAS procedure.

In [8] Zoph introduces two different kind of cells: a normal cell and a reduction cell. While the normal cell maintains the dimensionality of the input, the latter reduces the spatial dimension by a factor of 2. The final architecture is then built by stacking these cells in a predefined manner. The main advantages of choosing a cell-based search space are:

- Drastically lower size of the search space, as the cells usually consist of significantly less layers than whole architectures.
- Higher transferability of the architectures, as it is possible to adapt them to other dataset simply changing the number of cells used.

Consequently, this cell-based search space is also successfully employed by many other works. DARTS [9] by Liu *et al.* is a significant example of the use of the cell-based search space and takes it a step further by introducing the concept of differentiable architecture research. Differentiable Architecture Search is a technique for finding neural network architectures using gradient descent. It involves relaxing the search space of possible architectures into a continuous space, making it differentiable, so that gradients can be calculated and used to optimize the performance of the network. In this approach, the architecture is represented as a Directed Acyclic Graph (DAG) of operations, and the search algorithm is able to learn the structure of the network, the order and type of operations, and the parameters of each operation. The result is a neural network architecture that is tailored to the specific task at hand and has the potential to achieve superior performance compared to hand-designed architectures.

This method does away with the need for labor-intensive and expensive computational architecture search techniques like reinforcement learning or evolutionary algorithms. Furthermore, the differentiability of the architecture search enables the

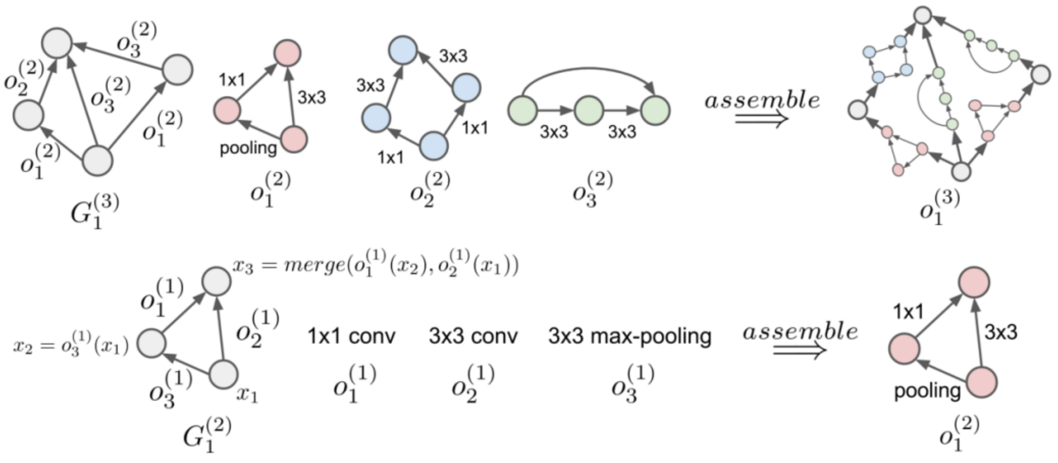

Figure 3.5: An example of a three-level hierarchical architecture representation is shown. The level-2 motif is assembled using level-1 primitive operations, as seen in the bottom row. The level-3 motif that is created by combining level-2 motifs is displayed in the top row. Source:[10]

use of large amounts of labelled data, allowing for the discovery of more accurate and efficient architectures for tasks such as image classification, object detection, and semantic segmentation. Overall, DARTS has contributed to the development of neural architecture search, and its methods have been widely used and enhanced in subsequent works.

Hierarchical Search Space When using a cell-based search space, a new design choice emerges, namely how to select the macro-architecture. Ideally, both the macro-architecture and the micro-architecture (cell structure) should be optimised together. Otherwise, once a high-performing cell has been identified, one is forced to manually engineer the macro-architecture. One step in the direction of optimizing macro-architectures is the hierarchical search space introduced by Liu *et al.* [10]. The hierarchical search space can be seen as a multi-level representation of the architecture, and the search process can be performed either by reinforcement learning, evolutionary algorithms, or gradient-based methods. The process starts with a small set of primitives, such as convolutional and pooling operations, at the lower level of the hierarchy. Then the higher-level computational graphs, or motifs, are formed using the lower-level motifs as building blocks. The motifs at the top of the hierarchy are stacked several times to form the final neural network. This approach allows search algorithms to implement powerful hierarchical modules in which any changes in the motifs are propagated to the entire network immediately.

Consider a hierarchy of L levels where the $l-th$ level contains M_l motifs.

The highest-level $l = L$ contains only a single motif corresponding to the full architecture, and the lowest level $l = 1$ is the set of primitive operations.
We recursively define $o_m^{(l)}$, the $m - th$ motif in level l, as the composition of lower-level motifs $o^{(l-1)} = \left\{o_1^{(l-1)}, o_2^{(l-1)}, ..., o_{M_{(l-1)}}^{(l-1)}\right\}$ according to its network structure $G_m^{(l)}$:

$$o_m^{(l)} = assemble\left(G_m^{(l)}, o^{(l-1)}\right)$$

A hierarchical architecture representation is therefore defined by:

$$\left(\left\{\left\{G_m^{(l)}\right\}_{m=1}^{M_l}\right\}_{l=2}^{L}, o^{(1)}\right)$$

as it is determined by network structures of motifs at all levels and the set of bottom-level primitives. An actual illustration of a hierarchical space can be seen in figure 3.5. At first, there are only primitive operations that are 1x1 conv, 3x3 conv, 3x3 max-pooling and the network structure $G_1^{(2)}$. At the first step, the assemble operation is performed on the network structure and primitives listed above. Then, at each succeeding step, a new motif is created assembling the components of the previous motif.

The search space chosen has a significant impact on the accuracy and efficiency of the final model. The ideal search space should strike a balance between the architectures' expressiveness and the evaluation's computational expense. If the search space is too narrow, it may limit the ability of the algorithm to discover optimal architectures. If it is too broad, it may lead to excessive computational cost and difficulty in finding the best architecture. Therefore, choosing the right search space is crucial for the success of NAS.
The following section discusses Search Strategies that are well-suited for these types of search spaces.

3.3.2 Search Strategy

In Neural Architecture Search (NAS), a search strategy is a set of procedures and formulas for finding the best neural network architecture for a specific task. The search strategy determines how the search space, which contains possible architectures, is explored and which architectures are chosen to be evaluated. Many different search strategies can be used to explore the space of neural architectures, including Random Search and Grid Search, Bayesian optimization, Evolutionary methods, Reinforcement Learning (RL), and Gradient-based methods.

Random Search and Grid Search Both Random Search (RS) and Grid Search (GS) are simple search policies exploited by several NAS studies. Random search is certainly the most simplistic strategy that consists in sampling architectures randomly from the predefined search space. Random search has been shown to perform well in practice, especially for smaller search spaces, but it can be computationally expensive and can struggle to find the optimal architecture for more complex problems.Random search is usually used as a baseline for comparison with other search policies and algorithms in the majority of the papers ([11], [12]), and it has proven to be a simple but effective approach for optimising hyperparameters in deep neural networks. For what concerns Grid Search , it has been used as a search policy in NAS. However, it is strongly limited by the computational cost and time requirements of testing each combination. This method has been used in some early NAS papers, but it has largely been replaced by more efficient and effective search policies.

Bayesian Optimization It is the most popular search strategy in NAS. This approach uses Bayesian methods to model the uncertainty in the search process and guide the optimization towards the best architecture. The basic idea is to use a probabilistic model to estimate the performance of the architectures and to guide the search towards the most promising ones. The optimization process involves iteratively selecting the architectures to evaluate, based on the model predictions, and updating the model parameters with the newly acquired performance information.

Evolutionary Methods These methods are inspired by the theory of evolution and natural selection in biology and are used to generate new architectures by combining and mutating existing ones. The evolutionary algorithms can be effective in solving complex optimization problems like NAS, but they can be computationally expensive due to the large number of architectures that need to be evaluated. Evolutionary methods make use of Evolutionary algorithms (EA) that are population-based global optimizer for black-box functions. They consist of the following essential components: initialization, parent selection, recombination and mutation, survivor selection. The initialization involves fixing how the first generation of the population is generated. Then, until termination the following steps are done (Figure 3.6):

1. Select parents from the population for reproduction.
2. Apply recombination and mutation operations to create new individuals.
3. Evaluate the fitness of the new individuals.

4. Select the survivors of the population.

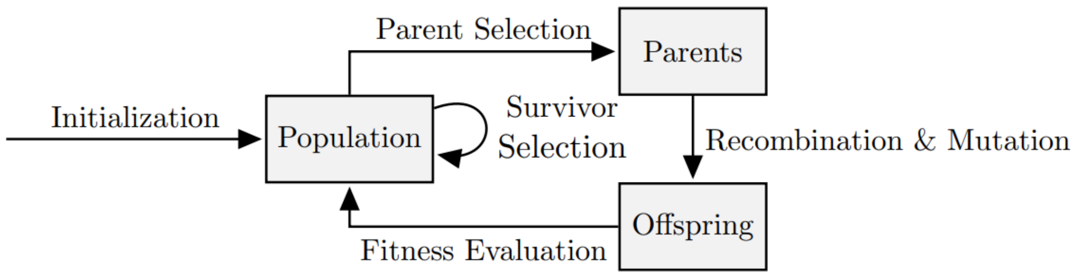

Figure 3.6: A general framework for evolutionary algorithms. Source:[13]

The population in the context of neural architecture search consists of a collection of network architectures. In the first step, either a parent architecture or a pair of architectures are chosen for mutation or recombination. The steps of mutation and recombination refer to procedures that produce novel architectural designs in the search space, which are then assessed for fitness in step 3 and iterated until the process is complete. In the field of NAS, mutation operators are frequently the only ones employed. There is no proof that a recombination procedure performed on two highly fit people would produce children with similar or higher fitness levels. The most common parent selection method in NAS is *tournament selection*. This method selects the individuals for further evolution in two steps. First, it selects k individuals from the population at random and then it iterates over them in descending order of their fitness while selecting individuals for further steps with some (presupposed) probability p.

Genetic algorithms are a class of evolutionary algorithms that have been extensively used in NAS. These methods are so-called because they use a *genotype*, a fixed-length encoding, to represent individuals in their methodology. On this representation, mutation and recombination operations are carried out. Furthermore, a genotype is used to create the physical appearance of the individual, which in the case of NAS is the architecture. The *phenotype* is the manifestation of this. It's crucial to remember that certain genotype components can be either active or inactive. If a change in the genotype causes a change in the phenotype, the information contained in the genotype is said to be active (given all other information remains the same).

Liu *et al.* [10] proposed an evolutionary algorithm that works in a hierarchical search space as described in 3.3.1. At every step, a mutation chooses the hierarchical level that will be altered and modifies the level's representations. The population starts out with 200 trivial genotypes, which are then diversified by applying 1000 random mutations. Furthermore, only 5% of the population is chosen as parents through tournament selection, and no one is eliminated during the evolutionary process. The mutation phase has the ability to modify, add, or remove operations from the genotype. The decision of the authors to use an evolutionary algorithm in conjunction with a hierarchical search space enables effective and efficient exploration of the vast search space.

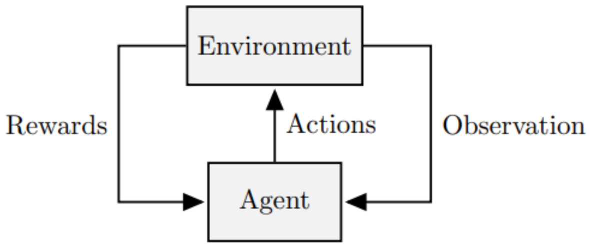

Figure 3.7: A general framework for reinforcement learning algorithms. Source:[13]

Reinforcement Learning RL Reinforcement learning (RL) approaches are useful for modelling a sequential decision-making process in which an agent interacts with its environment with the sole goal of maximising its future return. Figure 3.7 shows the generic framework for reinforcement learning algorithms and the functioning of this is described below. Through repeated interactions with the environment, the agent learns how to improve its behaviour. When the agent takes a specific action, at step t, it takes in exchange the observation of the state and a reward $r^{(t)}$. On the other hand, the environment responds to the agent's action by changing states and emitting the corresponding observation and reward. The goal of the agent is to maximise return, which is defined as the discounted sum of rewards over T decision steps.

$$\sum_{t=0}^{T} \gamma r^{(t)} \tag{3.1}$$

Naturally, such approaches are well suited for neural architecture search where the agent, namely the search algorithm, makes decisions to change the system's state, i.e. the architecture, in order to maximise the long-term objective of high performance which is frequently measured in terms of accuracy. Even in this situation, there are a lot of combinatorially possible options.

An important example of the application of RL is in the paper [7] by Zoph *et al.*. According to the authors, using RL to search for the best architecture can overcome some of the limitations of traditional methods such as Random Search and Grid Search and Bayesian Optimization, which can be time-consuming or limited in their ability to explore the search space. Moreover, the paper specifically describes a technique for searching through the architecture space using a reinforcement learning agent that has been trained to maximise a reward signal reflecting how well the architecture performed on a validation set. The architecture is represented as a series of operations, which the agent chooses one at a time until the final architecture is produced. The proposed method was evaluated experimentally by the authors, who demonstrated that it performs better than other NAS methods and conventional search techniques.

Gradient Descent-based Optimization Gradient Descent-based (GD) optimisation methods differ from previous search policies because they do not sample the neural network architecture from a discrete search space. DARTS algorithm [9] was the trailblazer in searching for neural architectures over a continuous and differentiable search space, with a GD-based approach. Specifically, Liu *et al.* proposed searching over a cell-based search space, performing a continuous relaxation to enable direct gradient-based optimization:

Let O be the set of all the candidate operations, where each operation represents some function $o(\cdot)$ to be applied to $x^{(i)}$. The choice of a particular operation is relaxed as shown in equation 3.2.To make the search space continuous, they relax the categorical choice of a particular operation to a *Softmax* over all possible operations:

$$\overline{o}^{(i,j)}\left(x\right) = \sum_{o \in O} \frac{exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o' \in O} exp\left(\alpha_{o'}^{(i,j)}\right)} o\left(x\right) \tag{3.2}$$

where the operation mixing weights for a pair of nodes (i, j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|O|$.

After the relaxation, the task of finding architectures is transformed into a joint optimization of the neural architecture and its weights. By continuously relaxing the search space, this method improves search efficiency, making it the fastest, best-performing, and most widely used method.

In summary, NAS employs a variety of search strategies, including such random search, grid search, Bayesian optimization, evolutionary algorithms, gradient-based optimization, and reinforcement learning. Each strategy has advantages and disadvantages, and the selection of a search strategy is largely influenced by the particular issue at hand.

3.3.3 Performance Estimation Strategy

A key component in Neural Architecture Search (NAS) algorithms is the Performance Estimation Strategy, which is responsible for evaluating the performance of candidate neural network architectures. To guide the search process of the neural network architecture, the performance estimation strategy need to estimate the performance of a given architecture A they consider.

Some of the commonly used performance estimation strategies in NAS are:

- Full training: This involves training each candidate architecture from scratch on the entire training dataset to estimate its performance. While this method provides accurate performance estimates, it can be very computationally expensive, especially for large datasets and complex architectures An intriguing application of full training as performance estimation strategy is in [14] by Chen *et al.* Their approach consists in dividing the search process into a number of stages, increasing the depth of the neural network at the conclusion of each stage, to reduce the cost of the process. In this manner, the full training is carried out first on smaller networks, taking less time, and on larger networks only if considered necessary.

- Early stopping: It refers to stopping the training of a neural network when the validation error stops improving, to avoid overfitting. This method is less computationally expensive than full training, but it may not provide accurate estimates, particularly for architectures that require more training time. In ENAS [15], Pham *et al.* exploit this performance estimation strategy so that the neural network is trained for a fixed number of epochs, and the validation accuracy is used as an estimate of the test accuracy.

- Surrogate models: In NAS, surrogate models are used as a means of estimating the performance of neural architectures without training them from scratch. These models learn to predict the accuracy of a given architecture based on a smaller set of trained architectures. However, the accuracy of the performance estimates depends on the quality of the surrogate model. One popular type of surrogate model is the Gaussian process (GP), a probabilistic model that can capture the uncertainty in the performance estimates, and can provide information about which regions of the search space are most promising. This has been used in several NAS methods, such as SMBO (Sequential Model-Based Optimization) in [16] and BOHB (Bayesian Optimization and Hyper-band)[17] .

  Another type of surrogate model used in NAS is the neural architecture predictor, which is typically trained to predict the accuracy of neural architectures based on their architectural descriptions. One example is DARTS [9], which uses a predictor to estimate the validation accuracy of candidate architectures during the search process.

- Transfer learning: It is a machine learning technique in which a model trained on one task is re-used as a starting point for training a model on a different but related task. In the context of neural architecture search (NAS), transfer learning can be used to speed up the search process by transferring knowledge learned from one architecture search to another. This approach can be particularly useful when searching for architectures in similar domains or when the search space is hierarchical. However, the accuracy of the performance estimates depends on the similarity of the architectures and the quality of the initial set.

Speed-up the Performance Estimation The simplest performance estimation strategy consists in fully training of the neural architecture A on training data and evaluating its performance on validation data. However, as previously said, this can yields computational demands in the order of thousands of GPU days for NAS. The most notable example of how long and complex the process is, is in [7] where Zoph and Le used 800 GPUs and three up to four days to train and test the selected neural network architectures.

Thus, the need to develop methods for accelerating performance estimation arose inescapably. Here are some of the most used.

- One-Shot Models, Weight Sharing : One-shot neural architecture search (NAS) is a class of NAS techniques that employs a single neural network to represent and look for architectures across the entire architecture space. This is accomplished by training a super network, which is a large neural network made up of many smaller sub-networks, with architectural parameters that determine which parts of the super network are used. Then, using gradient descent, the parameters of each sub-network are optimised to minimise a loss function that incorporates the training loss as well as an architectural regularisation term that promotes the use of fewer parameters.To effectively explore the space of architectures, one-shot NAS methods employ the weight sharing technique. Weight sharing allows sub-networks to share information and reduce the number of unique weight tensors that need to be optimised by tying the weights of the sub-networks to those of the super network during the optimization process. As a result, searching across the space of architectures costs less computationally. One example is given in ENAS [15], by Pham *et al.*. In ENAS, architecture descriptions are generated by a recurrent neural network (RNN), and a set of parameters that are common to all architectures is learned using gradient descent. In comparison to conventional methods, ENAS is able to drastically reduce the computational cost of NAS by utilising weight sharing and a shared parameterization.
- Network Morphism : In network morphism, new neural network architectures that are topologically related to an existing architecture, are discovered, but with different numbers of layers, widths, or filter sizes. Finding networks that

perform similarly to existing networks but use less resources can be helpful in the search for neural architectures that are resource- and computation-efficient.

- Multi-fidelity Optimization : Multi-fidelity optimization is a group of methods that are frequently employed in the context of neural architecture search (NAS) to expedite the search process by assessing candidate architectures incorporating evaluations at various levels of accuracy or computational cost. In the context of Neural Architecture Search (NAS), multi-fidelity methods typically involve using low-fidelity approximations or surrogate models to guide the search towards promising regions of the search space and then using high-fidelity evaluations to refine the search in those regions. This can significantly improve the efficiency of the search process, as it allows the algorithm to explore a larger portion of the search space in less time. Low-fidelity approximations are generally faster but less accurate. These methods usually involve evaluating the neural architecture's performance on a smaller and simpler dataset or using a simplified evaluation method. The objective of using these methods is to quickly discard poor-performing architectures and prioritize the search on the most promising ones. On the other hand, high-fidelity approximations are slower and more accurate and are used to refine the search within promising regions. In PDARTS [14] by Chen *et al.* we are given an example of the use of multi-fidelity approximations that led to a significant speed-up of the process and a reduction in computational cost, while still producing good results for the chosen architecture.

The previous sections discussed the NAS search space, search strategy, and performance estimation techniques. It could be argued that the search space and its selection are critical components of NAS because they can lead to the discovery of optimal architectures capable of achieving cutting-edge performance. Moreover, in order to efficiently explore the chosen search space, NAS search strategies have used a variety of techniques including random search, grid search, Bayesian optimisation, Evolutionary Algorithms, and Reinforcement Learning. Still, techniques like Early Stopping, Full Training, Early Stopping, Surrogate models can be used to precisely estimate the performance of architectures. To further accelerate the architecture search process, low-fidelity and multi-fidelity optimisation techniques have been developed, together with one-shot models and weight sharing.

Among the various types of low-fidelity optimization techniques, Zero-cost proxies are becoming increasingly popular as a performance estimation strategy in NAS. By employing a proxy metric that is easy to calculate, they can be used as a method to lower the computational cost of evaluating the performance of various architectures. More details are provided in the following section.

Chapter 4

Related Works

The objective of this chapter is to provide an overview of several studies that have investigated the development of efficient methods for neural architecture search (NAS) using zero-cost proxies (ZCs). To achieve this, we will first analyze the different types of zero-cost proxies and their related taxonomy. Then, we will review related works that have explored zero-cost proxies, highlighting the similarities and differences between them. Furthermore, the chapter will focus on two key components of this research, namely NASLib and Nas-Bench-201.

4.1 Zero-Cost Proxies in NAS

Over the past few years, there has been a growing interest in Neural Architecture Search (NAS) as a means to automate the search for optimal neural network architectures. However, this search is very expensive, particularly in terms of computational resources. Many NAS techniques, indeed, necessitate substantial GPU-hours to complete the search and the evaluation of every considered neural network architecture, mainly because the training phase can take day, weeks or even months to be completed. As a potential solution to this limitation, Zero-cost proxies (ZC) have been proposed.

Zero-cost (ZC) proxies in NAS are metrics that can be used to estimate the performance of a neural network architecture without actually training it from scratch. They are a type of low-fidelity performance estimation strategy that works exploiting a proxy metric that is simple to calculate, such as the number of parameters in the network or the number of floating point operations required for a forward pass. The Zero-cost (ZC) proxy can be categorized into different types, including:

- Data-independent ZC proxy : these measures entirely ignore the dataset, or use it only to set dimensions. They can be used on their own or in conjunction with other NAS methods to universally bias a search space before fine-tuning to a specific task.There is a natural baseline between data-independent ZC proxy, the number of `params`. This merely counts the number of weights in the network.

- Data-dependent ZC proxy : this includes ZC proxies that compute scores based on the data, but do not update the network weights with gradients. The vast majority of ZC proxies is data-dependent, including some measures motivated by theory, like `grasp`, and a number of heuristic measures (`grad-norm`, `snip`, `jacov`, `fisher`).

One of the first ZC proxies proposed wanted to estimate the separability of the minibatch of data into different linear regions of the output space [2]. Many other ZC proxies have been proposed since then, including data-independent ZC proxies [3] , [18], ZC proxies inspired by pruning-at-initialization techniques [19], [18], and ZC proxies inspired by neural tangent kernels [20].

EcoNAS In EcoNAS [21], the strategy adopted involves training and evaluating neural network architectures *under proxies*, *i.e* computationally reduced setting. There are four reduction factors: the number of channels for CNNs, the resolution of input images, the number of training epochs and the smaller number of data samples. The authors used the Spearman correlation coefficient as a parameter to assess the reliability of reduced settings; the correlation then measures the degree dependency between the original and reduced settings. When tested on the custom dataset of the authors, consisting of 50 neural architectures, the proxies were found to be highly effective, as evidenced by a strong Spearman correlation value. Additionally, the proxies demonstrated remarkable computational speed. For instance, it was observed that training the model for only $\frac{1}{10}$ of the total epochs was already sufficient to derive a good measure of neural network performance.

However, when the same proxies were utilized on a larger scale search space as NAS-Bench-201 [22], which comprises over 15,000 models, the correlation value was considerably lower. As a result, it was necessary to try different approaches and change the proxy literature.

4.2 Zero-Cost Proxies for Lightweight NAS

The paper by Abdelfattah *et al.* (2021) [3] represents the main work on the current Zero-cost proxies that are used in NAS. The authors aimed to develop proxies that could surpass the limitations of their predecessors, *i.e.* the reduced settings in EcoNAS, while maintaining the same level of computational efficiency. To achieve this, the pruning-at-initialization literature have been widely used within NAS.

A first example of the use of pruning-at-initialisation literature in NAS is AtomNAS by Mei *et al.* [23]. They proposed a dynamic network pruning technique to prune atomic blocks with negligible influence. An *atomic block* is a commonly used building structure composed of : convolution - channel-wise operation - convolution. For making the search space exploration easier, atomic blocks are pruned if they have a low importance for the entire network structure. AtomNAS is able to learn simultaneously the parameters of the network and the *weight* of the atomic block. At the end of the learning phase, atomic blocks with very small weights are pruned.

The principal aspect of pruning-at-initialization literature entails feeding a neural network with a single mini-batch of data and computing the *saliency* per parameter by multiplying the gradient and the initial parameter value. Saliency in the literature on pruning-at-initialization describes the significance of specific weights or connections in a neural network. The idea behind calculating saliency is to recognise the critical connections in a neural network right after initialization and then cut out the unnecessary ones.

The work by Abdelfattah *et al.* proposes six new zero-cost proxies, drawing from the recent pruning-at-initialisation literature, to rank neural network models in NAS. It then compares the new proxies with conventional proxies (EcoNAS) showing that Synflow performs better than the others. To validate the effectiveness of the proposed new metrics, the authors tested them on four other datasets with different sizes and tasks. The results showed that of the six zero-cost metrics initially considered, only Synflow demonstrated robustness across the different datasets and also when integrated within NAS algorithms.The work of Abdelfattah *et al.* lays the groundwork for additional Zero-cost proxies that can increase the accessibility of NAS without requiring computationally intensive resources. Future research should focus on determining why the Synflow ZC proxy is the best and most generally applicable.

Snip, Grasp and Synflow Three of the six Zero-cost proxies used by Abdelfattah *et al.* in their work are Snip [19], Grasp [24], and SynFlow [18]. All these methods originated for pruning neural networks and have recently been adapted for NAS. They share the common goal of identifying and removing unimportant connections or neurons in the network to increase efficiency and performance. All three methods use a metric to estimate the importance of each connection or neuron, the *Saliency*. In Snip [19] Lee *et al.* use the saliency as approximation of the changes in loss when a specific parameter is removed, instead, Wang *et al.* in [24] Grasp attempted to improve on the snip metric by approximating the change in gradient norm, instead of loss, when a parameter is pruned. Synflow [18] is actually a modified version of snip and grasp. Instead of using a minibatch of training data and cross-entropy loss(as in snip or grasp), since Synflow is a Data-Independent ZC proxy, the loss is simply calculated as the product of all the parameters of the network, therefore, no data is needed to calculate the loss or the Synflow metric. Below it is presented the Synflow formula used by [3] to score the neural networks in the Nas-Bench-201.

$$\texttt{synflow} : S_p(\theta) = \frac{\delta L}{\delta \theta} \odot \theta \tag{4.1}$$

where L is the loss function of a neural network with parameters θ, S_p is the per-parameter saliency and $\odot$ is the Hadamard product. In the end the saliency score is extended to the entire neural network by summing over all parameters N in the model: $S_n = \sum_i^N S_p(\theta)_i$

4.3 FreeREA : Training-Free Evolution-based Architecture Search

The work by Cavagnero *et al.* (2023) aims to provide a fast and efficient method for identifying neural networks that achieve high accuracy while preserving the typical constraints of tiny devices [1] . FreeREA is designed to be completely training-free, allowing for a significant reduction in computation time and resource requirements and maintaining the limitations of small devices.

The paper presents their approach to neural architecture search (NAS) that involves using the proxies as training-free metrics to evaluate the performance of candidate architectures and using an evolution approach to search for the optimal model, in the constrained scenario of tiny models. To evaluate the suitability of various metrics as a substitute for test accuracy, the authors conducted a series of performance tests on NATS-Bench [25]. Their findings indicate that LogSynflow (paragraph 4.3), Linear Regions, and the number of Skipped Layers are the most reliable metrics.

Additionally, Cavagnero *et al.* pioneered the concept of combining these metrics to estimate neural architecture performance. They achieved superior empirical results by summing the normalized scores of each individual metric, rather than relying on a standard cumulative classification score

With regard to the search policy, the authors use an improved version of Tournament Selection with Ageing (REA) [26]. The novelty introduced concerns the sample phase of this genetic algorithm where, in order to improve the search capability, two parents are sampled for each step. The two parents are then independently mutated to generate two children, while a third child is generated through a crossover operation. This increases the exploration capability of the search algorithm since, while mutation is inherently local, the combination of two different genotypes can lead to more varied individuals which significantly differ from their parents.

The authors conducted experiments on two benchmarks, NATS-Bench [25] and NASBench-101 [27], to evaluate their method, FreeREA. Interestingly, the results of the experiments demonstrated that FreeREA not only efficiently searches for neural architectures, without training any candidates, but also outperforms state-of-the-art training-based methods, making it the most efficient and accurate NAS algorithm currently available for the considered benchmarks. Additionally, the search time for FreeREA is four orders of magnitude shorter than that of the state-of-the-art algorithms.

LogSynflow LogSynflow was introduced by Cavagnero *et al.*, grew out as a revision of Synflow. The latter started out as a tool for a local evaluation of the relevance of the weights in the network and was later extended to allow scoring of the entire neural network. However, to calculate Synflow we are very likely to get a gradient explosion, even in small architectures. Consequently, Synflow might overlook the importance of the values of the weights because the term associated with the gradients can be several orders of magnitude higher than the corresponding weight.

LogSynflow proposes to scale the values of the gradients with a logarithmic function before summing the contributions of each weight in the network, so as to restore the correct weight values. Below it is presented the LogSynflow formula:

$$\texttt{logsynflow } : S_p(\theta) = \theta \cdot log\left(\frac{\delta L}{\delta \theta} + 1\right) \tag{4.2}$$

The performance of LogSynflow is a significant aspect to consider, as observed by the authors. They tested the ZC proxy on three datasets of NATS-Bench [25], which is an improved version of Nas-bench-201 benchmark, and achieved high correlation values.

Hence, a direct comparison of the results obtained by the authors with those of this thesis work is currently not possible.

4.4 NAS-Bench-Suite-Zero

The paper by Krishnakumar *et al.* (2022) introduces a comprehensive dataset of Zero-cost proxies, for accelerating research on Neural Architecture Search (NAS) [28]. The authors evaluated 13 ZC proxies on a total of 28 tasks through NASLib [29], yielding by far the most comprehensive dataset and single code base for ZC proxies. As a result, the experiments on ZC proxies could be conducted more quickly and without ambiguity due to different implementations or training pipelines.

The starting point for this work is the notion that, while ZC proxies hold a lot of promise, they are still understudied in some ways and their true potential has yet to be realised. In particular, it is almost unknown what outcomes can be obtained by incorporating ZC proxies into NAS algorithms. In addition to providing the most comprehensive dataset of ZC proxies, it aims to answer questions related to generalizability, complementariness and bias in ZC proxies. Based on the results obtained by the authors, it can be inferred that the `nwot` ZC proxy exhibits the highest Spearman correlation among the benchmarks. However, it should be noted that its performance is not entirely consistent across all of them. For instance, in the case of TransNAS-Bench-101 [30] , all the ZC proxies display unsatisfactory results, whereas most of them perform well on the NAS-Bench-201 benchmark. Thus, it can be concluded that only a handful of ZC proxies demonstrate the ability to generalize effectively across different tasks and benchmarks. Nonetheless, the authors propose to exploit this feature to assess the similarities between the various benchmarks in meta-learning settings.

The authors arrived at another significant conclusion which sets the groundwork for the possibility of combining multiple ZC proxies. They determined the level of complementary information that each ZC proxy can offer, which they found to be closely linked to the benchmark type. While this may initially appear to be a drawback, the authors propose that a machine learning model could effectively identify valuable combinations of ZC proxies, opening up a new perspective regarding the use of ZC proxies.

This work unquestionably marks a turning point in the investigation of ZC proxies and the search for new viewpoints on their application in NAS. However, although this work has made substantial progress in motivating and increasing the speed of research on ZC proxies, there are still some limitations. The authors of this work have only presented an empirical analysis, which is a significant limitation. While they do provide an initial theoretical assessment in the appendix, they fail to provide clear conclusions. Additionally, the idea of combining several proxies together is also limited. The authors note that there is a substantial complementarity of information between the ZC proxies in some benchmarks, but this degree of complementarity depends on the specific NAS benchmark. This implies that combining proxies based on benchmark-dependent features may not always provide the right information. Therefore, a more intrinsic feature of the ZC proxy itself should be identified for combining proxies. Machine learning models could be useful in identifying such features and finding useful combinations of ZC proxies.

Nwot Krishnakumar *et al.* have identified Nwot as the top-performing and most consistent proxy among all the benchmarks tested [28]. `nwot`, that stays for *NAS without training* was introduced by Mellor et al. (2020) in [2]. Nwot is actually the latest version of the metric proposed by Mellor *et al.* Its earlier version, known as

Jacov, has also been widely studied in the literature. This is one of the earliest work we can find that attempts to perform a type of NAS by scoring neural networks, without training them, but computing statistics from a forward pass of a single mini-batch of data. The Nwot proxy captures the activation correlation within a neural network when it is subjected to different inputs within a mini-batch of data. One of the key practical benefits of using Nwot is its rapid execution time, which can be crucial when running the NAS process multiple times for different datasets. This makes Nwot an attractive option for researchers and practitioners who need to optimize the efficiency of their NAS algorithms.

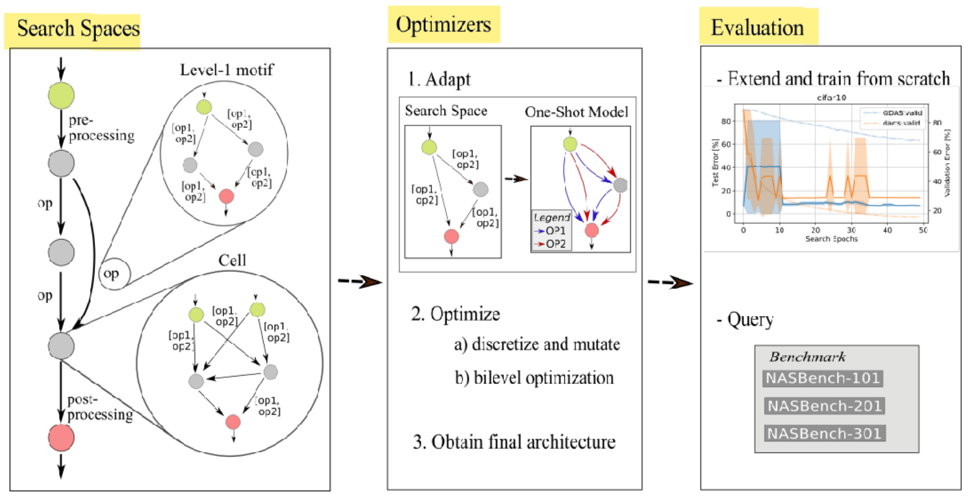

Figure 4.1: A high-level representation of the NASLib library which follows the typical steps of NAS. There are three main blocks that interacts each other : Search spaces, Optimizers and Trainers/Evaluators. Source:[29]

4.5 NASLib : a NAS library built upon PyTorch

NASLib by Ruchte *et al.*, is a new open-source library designed to facilitate neural architecture search (NAS) by providing a highly modular and flexible framework for researchers to develop and evaluate NAS algorithms [29] . With its features and user-friendly interface, NASLib allows users to easily experiment with a variety of search spaces, search strategies and performance estimation strategies, enabling them to discover optimal neural architectures for a wide range of tasks. The authors state that NASLib and its implementation were developed in response to a specific need: the necessity of a library that could provide basic strategies without requiring researchers to implement them. This frees up time and mental resources that can be dedicated for developing new and innovative ideas.

Figure 4.1 shows a high-level representation of the NASLib library and its approach to neural architecture search. First, the search space is defined, representing the set of potential architectures that the framework will consider. Next, the optimiser begins its search, aiming to identify the optimal architecture within the defined search space. Finally, the selected architecture is thoroughly evaluated to determine its performance.

A minimal example on how running the search and evaluation for DARTS [9] using NASLib is provided in the following snippet of code.

```python
from naslib.search_spaces import DartsSearchSpace
from naslib.optimizers import DARTSOptimizer
from naslib.defaults.trainer import Trainer

config = utils.get_config_from_args()

search_space = DartsSearchSpace()
optimizer = DARTSOptimizer(config)

optimizer.adapt_search_space(search_space)
trainer = Trainer(optimizer, config)
trainer.search()
trainer.evaluate()
```

NASLib relies on an effective representation of the search space to facilitate the combination of optimizers and search spaces. In the field of Neural Architecture Search (NAS), the most common way of representing search spaces is through Directed Acyclic Graph (DAG), which can be easily implemented as computational graphs in PyTorch. The search space in NASLib is built upon NetworkX [31] (Hagberg et al., 2008). This allows the complete definition of the search space in a single location, using recursive calls of the same object. This unified representation allows both high-level abstractions, such as the macro-graph, and low-level details, such as operations on edges and/or nodes of the Directed Acyclic graph (DAG), to be defined. One of the primary goals of NASLib is to promote the re-usability of search methods across different search spaces, thereby reducing the overhead involved in constructing search spaces from scratch.

In addition to its capabilities for graph creation and manipulation, NASLib offers a unified interface for several Neural Architecture Search (NAS) benchmarks. Currently, the following benchmarks are supported:

- NAS-Bench-201 [22] : It is a benchmark for evaluating NAS algorithms. It consists of a collection of 15.625 unique convolutional neural network (CNN) architectures. The architectures are evaluated on three datasets: CIFAR10, CIFAR100 and ImageNet16-120.

- NAS-Bench-101 [27] :This benchmark consists of a set of 4.096 neural network architectures, each trained and evaluated only on the CIFAR-10 dataset.

- NAS-Bench-301 [32] : It consists of a set of 10.000 neural network architectures that are trained and evaluated only on CIFAR10.

- NAS-Bench-ASR [33] : It is a benchmark for evaluating NAS algorithms for Automatic Speech Recognition (ASR) tasks. It consists of a set of 5.000 unique convolutional and recurrent neural network (CNN and RNN) architectures, each trained and evaluated on the Google Speech Commands dataset.

- NAS-Bench-NLP [34] : This benchmark is for evaluating NAS algorithms for natural language processing (NLP) tasks. It consists of a set of 8.840 neural network architectures, each trained and evaluated on the WikiText-103 language modeling dataset.

- TransNAS-Bench-101 [30] : It is a benchmark for evaluating transfer learning performance of NAS algorithms. It is an extension of NAS-Bench-101 and consists of a set of 1.000 neural network architectures, each trained and evaluated on the ImageNet dataset using a fixed set of hyperparameters. In addition, each architecture is also fine-tuned on 7 computer vision tasks.

This functionality enables researchers to query results from these benchmarks at any time and to train discrete optimizers without needing to modify any code when switching between applications or changing the search space and corresponding benchmark API.

In this thesis work, experiments were performed on the NAS-Bench-201 benchmark because it provides a unified benchmark for the most up-to-date NAS algorithms, including all cell-based NAS methods.

4.6 NAS-Bench-201

The NAS-Bench-201 [22] is a neural architecture search (NAS) benchmark and search space. The architecture of each model inside the NAS-Bench-201 search space is defined by a stack of cells that fit into a predefined skeleton. By decomposing the architecture into smaller components, the task of finding an optimal architecture is simplified to finding a good cell.

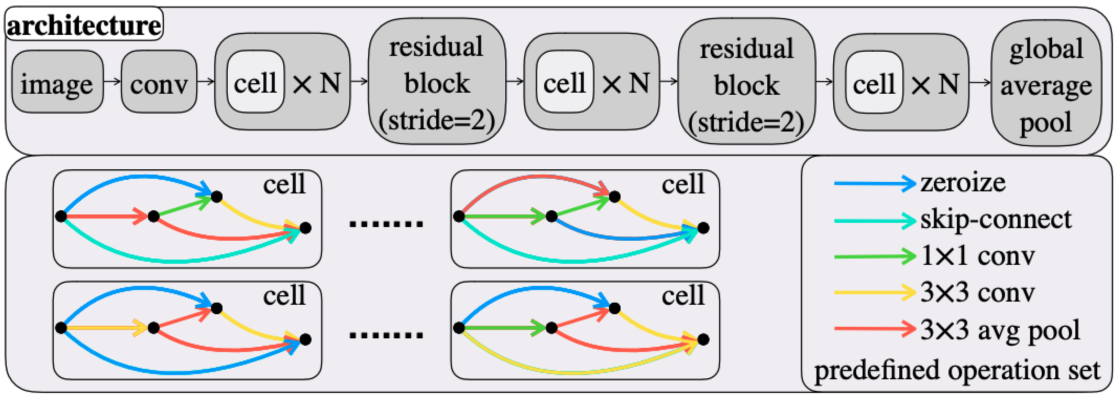

Figure 4.2: On the top the figure shows the macro-skeleton of each candidate architecture in the search space. To the bottom left, there are some examples of cells with 4 nodes. To the bottom right, the set of predefined operations that are used in our search space are displayed. Source:[22]

Architectures in the search space The design of the search space is based on the successful neural cell-based NAS algorithms introduced previously by Liu [9] et al. (2019), Zoph [8] et al. (2018), and Pham [15] et al. (2018). As illustrated in Figure 4.2, the search space consists of a macro-skeleton architecture that starts with a 3x3 convolution layer with 16 output channels and a batch normalization layer. The main body of the skeleton consists of three stacks of cells that are connected by a residual block with stride=2. Each cell is repeated $N = 5$ times and has 16, 32, and 64 output channels for the first, second, and third stages, respectively. At the end of the macro-skeleton architecture, it is included a global average pooling layer that flattens the feature map into a feature vector. Then a fully connected layer with a *Softmax* activation it is employed to transform the feature vector into the final prediction for classification.

For what concerns the searched cells, they are represented as a densely connected Directed Acyclic Graph (DAG). The DAG is obtained by assigning a direction from the i_{th} node to the j_{th} node, with $i < j$, for each edge of the graph. In this DAG, each edge is associated with an operation that transforms the feature map from the source node to the target node. Each DAG has $V = 4$ nodes, where each node is the sum of all feature maps transformed through the associated operations of the edges pointing to this node. The possible operation are shown in figure 4.2 on the bottom right. So, the set of the possible operation O has five operations :

$$\begin{aligned}\{&(1)\ \text{zeroize}, (2)\ \text{skip connection}, (3)\ \text{1-by-1 convolution},\\ &(4)\ \text{3-by-3 convolution}, (5)\ \text{3-by-3 average pooling}\}\end{aligned} \tag{4.3}$$

In this case, when speaking of convolution operation, the sequence of ReLU, Convolution and Batch Normalisation is implied.

Dataset The architectures in the NAS-Bench-201 search space are trained and tested on three widely-used datasets: CIFAR-10, CIFAR-100 [35], and ImageNet16-120 [36], chosen based on their popularity and relevance in the field of computer vision.

- CIFAR10 : It is a widely used image classification dataset, comprising 60.000 32×32 color images divided into 10 classes. The original training set contains 50.000 images, with 5.000 images per class. In order to have a separate validation set, the original training set was divided into two groups, each with 25.000 images and 10 classes. We designate the first group as the new training set and use it to train models, while the second group serves as the validation set for evaluating model performance.
- CIFAR100 : It is a variant of CIFAR-10, but with the fine-grained categorization of 100 classes. The original training set contains 50,000 images, while the original test set contains 10,000 images. The validation set is creating splitting the original test set into two groups of 5,000 images each.
- ImageNet16-120 : This is constructed from the down-sampled variant of ImageNet (ImageNet16×16). [36]. Indeed, images are down-sampled to 16×16 pixels to form ImageNet16×16, from which all images with label $\in [1, 120]$ to construct ImageNet16-120. At the end, this contains 151.700 training images, 3.000 validation images, and 3.000 test images with 120 classes.

Here is provided a code snippet demonstrating how to instantiate NAS-Bench-201 search space inside the NASLib library and how the primitive operation inside a cell are set. The nodes of the DAG are labeled starting *from 1 to 4* and then the edges are created such that $u, v \in \{1, 2, 3, 4\}$ with u < v. The output node is the node with the highest index, and the input nodes are those without an incoming edge. For each stage of the overall search space, different settings are needed when setting up primitive operations. For this, three different scopes are identified.

```
class NasBench201SearchSpace(Graph):
OPTIMIZER_SCOPE = [
"stage_1",
"stage_2",
"stage_3", ]

def __init__(self):
super().__init__()
# Cell definition
```

```
cell = Graph()
cell.name = "cell"
cell.add_nodes_from([1, 2, 3, 4])
cell.add_edges_densly()

# Set primitives as ops at the cells
channels = [16, 32, 64]
for c, scope in zip(channels,
self.OPTIMIZER_SCOPE):
self.update_edges(update_func=lambda current_edge_data:
                                    _set_cell_ops(current_edge_data,
                                    C=c), scope=scope,
                                    private_edge_data=True)
```

4.6.1 Querying NAS-Bench-201 through NASLib

In this thesis, we focus in particular on a remarkable feature of NASLib, namely the possibility to query NAS-Bench-201 and obtain information on the performance of models within its search space. This represents an excellent opportunity that provides access to over 15,000 models and gives us a preliminary knowledge of the accuracy obtained by those models on the reference dataset. This unique capability has significant potential for our research and increases our ability to achieve our ultimate goal.

Below is a code snippet provided to facilitate a clearer comprehension of how to query NAS-Bench-201. After installing NASLib in the environment using either `pip` or `conda`, the next step is to initialize the NAS-Bench-201 dataset. Once the dataset is initialized, the benchmark provides a function that allows the access to all the architectures available. Finally, by using the `query_architecture()` function, it is possible to obtain the performance metrics, i.e the accuracy, of the specific architecture you are interested in.

```

from naslib.data.datasets import NASBench201Dataset

# Initialize the dataset
nasbench201_dataset = NASBench201Dataset()

# Load the data
nasbench201_dataset.load()

# Get all architectures in the dataset
architectures = nasbench201_dataset.get_all_architectures()

# Print the number of architectures
```

```
print("Number of architectures in NAS-Bench-201 dataset: ", len(architectures))

# Query performance metrics for a specific architecture
architecture = architectures[0]  # Example: get the first architecture in the dataset
perf_metrics = nasbench201_dataset.query_architecture(architecture)

# Print the performance metrics
print("Performance metrics for architecture: ", architecture)
print("Test accuracy: ", perf_metrics["test_acc"])
```

Chapter 5

Methodology

In this chapter, the focus is on the main problem addressed in this thesis, namely the study and creation of the Hybrid Zero-cost proxy.

5.1 The Hybrid Zero-cost proxy

The development of the new Hybrid zero-cost proxy started with a careful study of the definition of Zero-cost proxies, with a focus on its classification. Through an in-depth analysis of the literature and existing research on the subject, we were able to identify the main characteristics and peculiarities of zero-cost proxies. As previously stated in section 4.1, zero-cost proxies are classified according to their data dependency as:

- Data-Independent;
- Data-Dependent;

Table 5.1 displays the proxies that were examined and analyzed in this thesis. Each existing ZC proxy belongs to a single category and possesses the unique characteristics and behaviour associated with that category. Consequently, the ZC proxy is able to provide the neural network architecture score based on its individual behaviour as well as the broader characteristics associated with its category.

The central idea proposed in this thesis is to develop a novel Hybrid ZC proxy that seamlessly integrates both key features of data dependency, thus combining the strengths of data-independent and data-dependent proxies. This innovative approach represents a significant advancement in performance estimation of neural architecture in NAS, and has the potential to overcome the limitations of existing proxy solutions. By leveraging the complementary nature of the two data dependency characteristics, the Hybrid ZC proxy can provide a more comprehensive and accurate score for approximating the neural network architecture behavior.

Zero-Cost Proxy	
Data-Independent (DI)	Data-Dependent (DD)
	`epe-nas` [37]
	`fisher` [38]
`synflow[18]`	`grad-norm`[3]
`logsynflow[1]`	`jacov`[2]
`zen[39]`	`grasp`[24]
`l2-norm[3]`	`nwot`[2]
	`plain`[3]
	`snip`[19]

Table 5.1: List of ZC proxies discussed in this work. The distinction is made between data-independent and data-dependent.

The strategy developed proposes a two-stage approach to calculate the final neural network architecture score, using the Hybrid Zero-cost proxy.

1. The first stage involves calculating the initial score of each neural architecture in the benchmark for each considered ZC proxies. Subsequently, the correlation between the scores and the model's accuracy is determined. The best ZC DI (Zero-Cost Data-Independent) and ZC DD (Zero-Cost Data-dependent) are then selected and normalized. Once normalized, the scores are ready for refinement in the second step

2. In the second stage, the Hybrid ZC proxy is calculated by leveraging the previously chosen ZC DI and ZC DD. The new score is derived from a weighted sum of two parameters, taking into consideration specific cases

During the course of this thesis, the process of combining the two scores was a key focus of study and discussion. Several tests were conducted to evaluate different methods of combining the two normalized scores. Among these methods, a weighted summation of the two scores emerged as the most promising approach. The formula for the analytical expression is the equation 5.4.

5.2 Evaluation of the Hybrid Zero-cost proxy

To perform the final evaluation of the proposed Hybrid ZC proxy, the correlation between the measurement obtained through this approach and the actual accuracy achieved by the neural network during testing, is calculated. In particular, two different correlation measures are calculated:

- The Spearman Correlation : It is a statistical measure used to assess the strength and direction of the relationship between two variables. The Spearman correlation coefficient is represented by the symbol ρ and can take on values in $[-1, 1]$. A value of 1 indicates a perfect positive correlation, where the two variables move in the same direction with the same magnitude. A value of 0 indicates no correlation between the two variables and a value of -1 indicated a relative correlation. To calculate the Spearman correlation coefficient, the ranks of the values for each variable are first determined. The differences d_i between these ranks are then calculated, and the sum of the squares of these differences is used to compute the Spearman correlation coefficient. The formula for computing the Spearman correlation coefficient is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{5.1}$$

- The Kendall tau Correlation : it is a non-parametric measure of the association between two variables. It is similar to the Spearman correlation since it is based on rank-order data, but it uses a different formula to calculate the correlation coefficient. Like the Spearman correlation, the Kendall tau correlation coefficient can take values in $[-1, 1]$. n_c is the number of concordant pairs and n_d is the number of discordant pairs.

$$\tau = \frac{n_c - n_d}{n \cdot (n - 1)/2} \tag{5.2}$$

Critique of the Evaluation of Zero-Cost proxies The evaluation method described above was originally developed by Abdelfattah in [3] and has since been widely adopted as the standard to evaluate the effectiveness of a Zero-cost proxy. However, during our experiments, we discovered that this method of evaluation has some limitations and inaccuracies. In addition to calculating the correlation between each Zero-cost proxy and the actual accuracy of the model, we also extracted the accuracy of the model judged to be better from that same ZC proxy. However, we observed that a high correlation measure, measured with Kendall Tau or Spearman, did not always indicate that the zero-cost proxy found a better architecture than other proxies with a lower correlation. This suggests that using the correlation measure alone may not be a sufficient metric to fully assess the effectiveness of zero-cost proxies. Other factors, such as the computational cost of each proxy, or the amount of information that it brings, may also be needed to better evaluate the performance of each proxy. Therefore, a more comprehensive evaluation metric that takes into account multiple factors may be necessary to accurately evaluate the efficacy of zero-cost proxies.

Given the limitations of using correlation as a sole metric for evaluating the effectiveness of the Zero-cost proxies, we decided to report in the results about our Hybrid ZC proxy, both the correlation value and the accuracy of the best-ranked model. Indeed, by reporting the correlation value, we can ZC proxies that use this metric as their primary evaluation tool. However, we also believe that it is important to report the accuracy of the best-ranked model, as this provides a more accurate assessment of the actual performance of the Hybrid ZC proxy. By reporting both metrics, we can provide a more comprehensive evaluation of each zero-cost proxy and enable a better comparison of the efficacy of different proxies.

5.3 Experiments

In this section, we provide a detailed description of the steps involved in the design and calculation of the new Hybrid ZC proxy: scoring of neural network architectures with existing ZC proxies, design of the Hybrid Zero-cost Proxy and evaluating the novel Hybrid ZC proxy.

All experiments were conducted exclusively on the Central Processing Unit (CPU), utilizing the Visual Studio Code development environment in conjunction with Anaconda, an open-source distribution of Python and R programming languages for scientific computing, data science, and machine learning tasks

5.3.1 Scoring neural network architectures with existing Zero-cost proxies

In the first step, we begin calculating the score of every neural network architecture in the NAS-Bench-201 search space using existing Zero-cost proxies proposed by previous works. We made the decision to perform these calculations for all models in the search space to ensure that no potential high-performing architectures were overlooked. By considering all 15.625 models, in the next steps, we can more comprehensively evaluate the quality of the newly proposed Hybrid Zero-cost proxy, against existing methods on the same set of models, providing a more meaningful comparison between methods. All the Zero-cost proxy metrics are listed in table 5.1. It should be noted that for each step, we perform three calculations for the following datasets: CIFAR10, CIFAR100, and ImageNet16-120.

In the following code snippet, we fix the configuration of the experiment, including the search space and the dataset to be tested. In the case presented, we set the search space to be NAS-Bench-201 and the dataset ImageNet16-120. Then we create the object `iterator_arcs` that enables iteration over the entire search space, allowing us to evaluate each architecture in NAS-Bench-201 one at a time.

```
#Load the configuration from the configs/predictor_config.yaml

config = utils.get_config_from_args()

#or set the searchspace and the dataset manually

config.search_space='nasbench201'
config.dataset='ImageNet16-120'
dataset_api = get_dataset_api('nasbench201', 'ImageNet16-120')

#Search Space creation
search_space = NasBench201SearchSpace()

#Get Itarator on archs
iterator_arcs = search_space.get_arch_iterator(dataset_api=
                                dataset_api)
```

After establishing the search space and dataset to be used, we extract each neural architecture accuracy score, i.e `actual_score`. To extrapolate the accuracy of each architecture on the test dataset, we query the NASLib library that contains all the accuracy scores for each search space and dataset. After obtaining the real accuracy of the evaluated neural network architecture, we then proceed to compute the scores for all the Zero-cost proxies that have been taken into account. We also record the time, which is measured in seconds, that was used to calculate each measurement.

```
# List of Zero-cost proxies considered
METRICS = ["synflow", "snip", "jacov", "grad_norm", "grasp",
"fisher", "epe_nas", "l2_norm", "logsynflow", "nwot", "plain", "
                                zen"]

def calculate_score_from_metric(metric, model, dataloader):
    start_time = time.time()
    predictor = ZeroCost(method_type=metric)
    score = predictor.query(model, dataloader=dataloader)
    end_time = time.time()
    return end_time-start_time, score

#Iterate over the neural network architectures
```

```
for i, op_indices in tqdm(enumerate(iterator_arcs)):
    ...

actual_score = model.query( metric=Metric.VAL_ACCURACY, dataset='
                                    ImageNet16-120', dataset_api=
                                    dataset_api)
output = list(
        map(functools.partial(calculate_score_from_metric, model=
                                    model, dataloader=test_loader),
                                    METRICS)
        )
    ...
```

5.3.2 Design of the Hybrid Zero-cost Proxy

The Hybrid Zero-cost proxy is essentially a combination of two zero-cost proxies, one independent of the data and the other dependent. The calculation of the proposed Hybrid Zero-cost proxy first involves a normalisation phase, then the selection of the best Data-independent (DI) and Data-dependent (DD) proxies and finally their combination.

Normalization of the scores To begin with, we retrieve the scores of each neural architecture that were previously calculated using the Zero-cost Proxies, as already described in section 5.3.1. To ensure consistency and comparability of scores, we employ a max-normalization process, resulting in improved empirical outcomes when assessed against a standardized cumulative ranking score. Working with normalized scores can be beneficial in several ways. First of all normalizing is particularly beneficial when analyzing or comparing data of different groups. Additionally, when examining the previously calculated scores, we notice important magnitude variations among them. This variability can be misleading and may result in inaccuracies when computing a composite score from these individual measures. In such cases, the relative importance of each measure in the composite score calculation may be biased towards the measure with the highest magnitude, leading to an incorrect final evaluation of the Zero-cost Hybrid proxy.

Select one Data-Independent and one Data-Dependent ZC proxy At this point, it is important to clarify the selection process of the two Zero-Cost Proxy measure used to obtain the Hybrid ZC proxy. As the Hybrid Zero-Cost Proxy approach involves combining one ZC DI proxy and one ZC DD proxy, we carefully evaluated and selected the best performing ZC proxy measures for both the categories. This ensured that we had an optimal combination of the two proxy methods for our experiments, which would help to improve the accuracy and reliability of our results. To this end, we computed the two correlation scores, namely the Kendall-Tau 5.2 and Spearman 5.1, in order to assess the efficacy of each proxy. After analyzing the results, we found that the `logsynflow` (4.3) was the best performing Data-Independent (DI) proxy, while the `nwot` (4.4) was the most effective Data-Dependent (DD) proxy.

Combination of LogSynflow (DI) and Nwot (DD) During the development of the thesis, the strategy for combining LogSynflow (DI) and Nwot (DD) has undergone numerous revisions and sparked intense debate. After careful consideration, we ultimately settled on a weighted summation approach for combining the scores, which occurs if and only if the normalized score of the Data-dependent ZC proxy is greater than the Data-independent one. This strategy was chosen after weighing various factors and considering input from multiple stakeholders. To provide greater clarity, we have compiled a summary of the key points that underpin this strategy:

- LogSynflow (DI) and Nwot (DD) are combined since they have demonstrated superior performance in both categories, as evidenced by their high Kendall-tau and Spearman correlation score.
- The strategy places a primary emphasis on the DI score, while utilizing the DD score in a supporting role. This is due in part to the fact that the DI score provides an evaluation of a model's capacity to handle inputs of a given size. However, it's worth noting that the DI score does not account for the specific data distribution at hand and thus cannot tailor the architecture judgement to the nuances of the data.
- In the event that the DI score misjudges the capability of a model, providing a low score, the DD score serves as a crucial support. By assessing the performance of a model using a mini-batch of data during the training process, the DD score provides additional information that can complement and refine the assessments made by the DI score. This dual approach helps ensure that our evaluations are as accurate and reliable as possible, ultimately leading to better model selection and performance.
- The two scores are combined using a weighted sum with alpha and beta parameters. The sum in performed element-wise only in case the DD Zero-cost proxy score is greater than or equal the DI Zero-cost proxy.

Below it is provided a clear and rigorous definition of the derivation process for calculating the hybrid ZC proxy.
Let $\vec{x}$ be the vector containing the n architectures of the benchmark we are analyzing,

then the Data-independent and Data-dependent ZC proxy normalized scores are defined as follow:

$$\begin{aligned}\vec{x} &= (x_1, x_2, \ldots, x_n)\\ DIN(x_i) = data\_independent\_normalized(x_i) &= \frac{logsyn(x_i)}{max(logsyn(\vec{x}))}\\ DDN(x_i) = data\_dependent\_normalized(x_i) &= \frac{nwot(x_i)}{max(nwot(\vec{x}))}\end{aligned} \tag{5.3}$$

The Hybrid zero-cost proxy can be conceptualized as a vector that is computed point-wise, employing the following function for its derivation:

$$h(x_i, x_i) = \begin{cases} \alpha DIN(x_i) + \beta DDN(x_i) & \text{if } DIN(x_i) \leq DDN(x_i) \\ DIN(x_i) & \text{if } DIN(x_i) > DDN(x_i) \end{cases}$$

$$\textbf{Hybrid ZC Proxy} = (h(x_1, x_1),\ h(x_2, x_2),\ \ldots\ h(x_n)) \tag{5.4}$$

5.3.3 Evaluating the novel Hybrid ZC proxy: calculation of the correlation values

Once the Hybrid zero-cost proxy formula has been derived, to assess the effectiveness of the new measure relative to previous ones, we calculate both Kendall-tau and Spearman correlation values between the Hybrid ZC proxy score and the actual Accuracy of the model. By comparing the correlation values of the new measure with those of previous ones, we can determine whether our new approach is superior and whether it warrants further consideration in future evaluations.

In addition to calculating Kendall-tau and Spearman correlations, we also conduct a study of the α and β parameters used in the weighted sum. Through this study, we are able to identify the values of α and β that lead to the highest levels of correlation scores. This analysis provides valuable insights into the weighting of the scores and enables us to fine-tune our approach for even greater accuracy and effectiveness. By refining the values of alpha and beta, we can achieve a higher final correlation, ultimately leading to better model selection and performance.

Chapter 6

Results

In this chapter, we present the results of our experiments to evaluate the effectiveness of the new Hybrid zero-cost proxy that we proposed in this Master's thesis. We will provide a detailed analysis of the results obtained for each of the three datasets that we analyzed, differentiating them according to the different phases of our experiments. It is also discussed the impact of alpha and beta parameters on the performance of our Hybrid Zero-cost proxy, providing additional insights into the relationship between these parameters and final correlation values. Finally, we conclude the chapter with a discussion of our findings and an exploration of the implications of our work for the field of NAS and, more in general, AutoML.

6.1 Scoring neural network architectures with existing ZC proxies

The first part of the experiment involved computing the score for each neural network architecture in the search space of NAS-Bench-201 using the currently available Zero-cost proxy metrics.

Each architecture's metrics took an average of 6.5 seconds to calculate, and the entire process took a total of 28 hours.

The following tables show the Kendall-tau and Spearman correlations computed for the three reference datasets: CIFAR10 (Table 6.2), CIFAR100 (Table 6.3), and ImageNet16-120 (Table 6.4). The tables are presented in the same order as the datasets were listed. The tables highlight the ZC proxies with the highest Kendall-tau and Spearman correlation values in bold, as obtained from our experiments. As part of our experimental methodology, we included a column in the table to report the Spearman correlation values calculated in NAS-Bench-Suite-Zero [28]. We deemed it important to include this column in order to assess the accreditation of our results, and to ensure that our ZC proxy measures are indeed reflective of the performance obtained by others in same setting.

During the process of comparing our correlation values to those reported in NAS-Bench-Suite-Zero, we did observe a difference in thousandths. We arrived at the conclusion that the observed differences are likely attributable to the variance in hardware settings between our system and the compared system. Nevertheless, these differences have minimal impact on the overall analysis. Therefore, we deemed our results to be acceptable and reflective of the actual performance of the models. It is worth noting that such minor differences are not uncommon in empirical studies and do not necessarily invalidate the findings, especially when they do not affect the main conclusions of the study.

Furthermore, it is important to note that the metric `logsynflow` is not included in the table as it is not calculated in the NAS-Bench-Suite-Zero benchmark. The evaluation of LogSynflow was carried out by the authors on a distinct search space, namely the NATS-Bench [25]. While the topology of the search space for NATS-Bench and NAS-Bench-201 is consistent, it is important to note that the results of the former benchmark were updated based on a larger number of evaluations for candidate architectures. Despite the lack of direct comparability with our results due to differences in the benchmarks used, we have chosen to include the results of LogSynflow as presented in FreeREA 4.3 in Table

LogSynflow	CIFAR10	CIFAR100	ImageNet16-120
Kendall-Tau	0,61	0,60	0,59
Spearman	0,81	0,79	0,78

Table 6.1: Kendall and Spearman correlation between LogSynflow and the test accuracy, evaluated on the three datasets of NATS-Bench [25].

The outcomes presented so far in this study reveal that the Zero-Cost (ZC) proxies with the highest Kendall-tau and Spearman correlation values are LogSynflow and Nwot for Data-Independent (DI) and Data-Dependent (DD) ZC proxies, respectively. Therefore, LogSynflow and Nwot are the most effective ZC proxies for accurately estimating the performance of neural network architectures on the three datasets examined.

Based on these results, in the next phase of the experiment, LogSynflow and Nwot are selected as the ZC DI and ZC DD proxies, respectively, for all three datasets. By doing so, we can ensure that the computation process of the novel Hybrid ZC proxy is driven by the most reliable and accurate ZC proxies available, which should lead to improved performance and efficiency of the neural network models.

	Kendall-Tau	Spearman	Spearman - NAS-Bench-Suite-Zero
Nwot	**0,58**	**0,77**	**0,77**
LogSynflow	**0,56**	**0,76**	-
Jacov	0,56	0,73	0,75
Synflow	0,54	0,73	0,73
EpeNas	0,54	0,72	0,7
L2	0,49	0,68	0,68
Snip	0,43	0,59	0,58
Grad	0,42	0,59	0,58
Grasp	0,33	0,48	0,51
Fisher	0,37	0,51	0,5
Zen	0,27	0,35	0,35
Plain	-0,12	-0,24	-0,26

Table 6.2: The table displays the Kendall-tau and Spearman correlations obtained from our evaluation of existing Zero-cost proxies on the CIFAR10 dataset. The results in the *Spearman - NAS-Bench-Suite-Zero* column are derived from the study conducted by [28].

	Kendall-Tau	Spearman	Spearman - NAS-Bench-Suite-Zero
Nwot	**0,62**	**0,80**	**0,80**
LogSynflow	**0,59**	**0,79**	-
Synflow	0,57	0,76	0,76
Jacov	0,54	0,71	0,71
L2	0,52	0,71	0,72
EpeNas	0,51	0,69	0,60
Snip	0,47	0,63	0,63
Grad	0,47	0,63	0,63
Fisher	0,40	0,55	0,54
Grasp	0,38	0,54	0,54
Zen	0,28	0,36	0,35
Plain	-0,16	-0,23	-0,21

Table 6.3: The table displays the Kendall-tau and Spearman correlations obtained from our evaluation of existing Zero-cost proxies on the CIFAR100 dataset. The results in the *Spearman - NAS-Bench-Suite-Zero* column are derived from the study conducted by [28].

	Kendall-Tau	Spearman	Spearman - NAS-Bench-Suite-Zero
Nwot	**0,60**	**0,78**	**0,77**
LogSynflow	**0,58**	**0,77**	-
Synflow	0,56	0,75	0,75
Jacov	0,54	0,71	0,71
L2	0,50	0,69	0,69
Snip	0,43	0,58	0,57
Grad	0,43	0,58	0,57
Grasp	0,39	0,55	0,55
Fisher	0,37	0,49	0,48
EpeNas	0,35	0,51	0,51
Zen	0,29	0,40	0,39
Plain	-0,15	-0,22	-0,22

Table 6.4: The table displays the Kendall-tau and Spearman correlations obtained from our evaluation of existing Zero-cost proxies on the ImageNet16-120 dataset. The results in the *Spearman - NAS-Bench-Suite-Zero* column are derived from the study conducted by [28].

6.2 Evaluating the new Hybrid Zero-Cost Proxy

To calculate and evaluate the new Hybrid Zero-Cost (ZC) proxy, we undertook a multi-step process. Each step was carefully designed and executed to ensure reliable and accurate results. In this section, we present the outcomes of the evaluation conducted on the Hybrid ZC proxy.

To evaluate the effectiveness of the newly proposed Hybrid Zero-cost proxy, we conducted experiments to measure the correlation between the proxy and the model accuracy. Specifically, we calculated both the Kendall-tau and Spearman correlations between the Hybrid ZC proxy and the test accuracy of the model, obtained by querying NASLib as previously explained in section 4.6.1. Furthermore, we provide the optimal values of the parameters α and β, which were determined through a rigorous evaluation process involving systematic variation of these two parameters over a predefined range of values of $[0, 100]$, step $= 0.5$. This critical stage of our experiments represents the final step in our evaluation.

Upon analyzing the results presented in tables 6.5, 6.6 and 6.7 it is evident that the newly proposed Hybrid ZC proxy exhibits consistent performances across all three datasets. The results demonstrate that the Hybrid ZC proxy outperforms LogSynflow and Nwot, in terms of obtaining higher correlations. This finding indicates that the Hybrid ZC proxy is a more effective tool for accurately evaluating

neural network architectures compared to the individual ZC proxies that compose it.
The results of our experiments confirm the assumptions made during the study. Specifically, we found that the Hybrid ZC proxy showed higher correlation values, which supports our initial assumption that combining the Zero-cost DI and Zero-cost DD proxies would provide more accurate estimates of neural network performance than using each proxy individually. Furthermore, our approach of using a weighted sum of the two proxies, only when the DD proxy provides more information than the DI proxy, was validated. Our hypothesis stated that the Hybrid ZC proxy should be calculated only when the DD Zero-cost proxy judges a network as performing better than its DI Zero-cost proxy counterpart. By incorporating additional information only when necessary, our combination approach enables us to be more precise and, at the same time, to save resources when the neural networks have already been judged as performing well and able to work with the considered input dimensions.

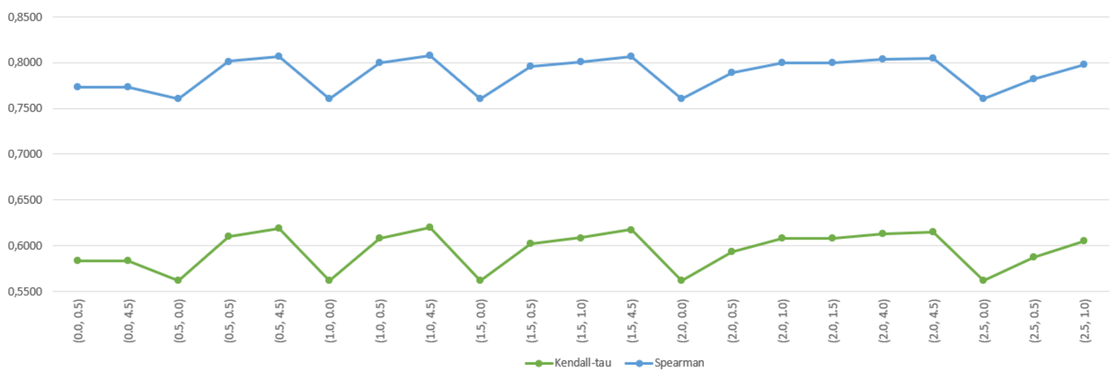

Figure 6.1: Kendall-Tau and Spearman correlation values of the Hybrid ZC proxies obtained varying the parameters α and β. The highest correlation values are obtained, for the first time, at the point where $\alpha = 0.5$ and $\beta = 1.5$.

Based on our results, we have found that the optimal values for the weighting parameters are $\alpha = 0.5$ and $\beta = 1.5$. It is important to note that these parameter settings represent the minimum values required to achieve the highest correlation values. Figure 6.1 illustrates how the Kendall-tau and Spearman correlation values vary as the alpha and beta parameters change. For the purpose of synthesizing information and enhancing graph readability, the complete range of variations in the two parameters is not depicted. Instead, we have chosen to highlight the segment that we find most illustrative. Although there is no clear trend observed, we can identify that the highest correlation values are achieved when alpha equals 0.5 and beta equals 1.5. While it is true that these values are attained at a later point, we have chosen to consider the point where the highest values are initially obtained as the points of the optimal parameters to use. Interestingly, we observed several cases where wider parameter settings did not lead to even higher correlation values. It is important to recognise that future investigations could benefit from a more in-depth exploration of the effects of wider parameter settings, whereby the analysis can gain a more comprehensive understanding of how weighting parameters may influence the performance of the proxy approach; we therefore encourage further research in this direction.

Overall, our results have demonstrated the effectiveness of the newly proposed Hybrid Zero-cost proxy as a tool for evaluating the performance of neural network architectures. By combining the strengths of its constituent ZC proxies, LogSynflow and Nwot, the Hybrid ZC proxy outperformed both individual proxies in terms of correlation values on the NAS-Bench-201 benchmark. Our findings provide a good starting point for further exploring the capabilities of the Hybrid ZC proxy and its potential applications in the neural network evaluation phase in NAS. As machine learning models continue to grow in complexity and size, the need for efficient and accurate evaluation methods becomes increasingly important. The Hybrid ZC proxy shows great promise as a tool for addressing this need, and we believe that further investigation into its capabilities and limitations can help to refine and optimize its use.

	Kendall-Tau	Spearman	Alpha	Beta	Top Accuracy
Hybrid ZC proxy	**0,62**	**0,81**	0,5	1,5	**90,74%**
Nwot	0,58	0,77	-	-	88,95%
LogSynflow	0,56	0,76	-	-	90,36%

Table 6.5: The table shows the Kendall-tau and Spearman correlation for Hybrid Zero-cost proxies on CIFAR10 dataset. Alpha and Beta here are the minimum values for which we obtain such results. Finally, the Top Accuracy is the accuracy value of the model found as optimal from that ZC proxy.

Final observation A final observation regarding the results obtained is that they serve as a practical realization of the points discussed in the previous section 5.2.

As previously noted, using the correlation, Kendall-Tau or Spearman, as the unique evaluation metric for Zero-cost proxies may not always be appropriate or fair. Specifically, our experiments with the CIFAR10 dataset revealed that the highest accuracy achieved by a model was 91,57% and the Hybrid ZC proxy was able to identify a model with a slightly lower accuracy of 90,74%, outperforming the other two ZC proxies, Nwot and LogSynflow, (which identified models of accuracy

	Kendall-Tau	Spearman	Alpha	Beta	Top Accuracy
Hybrid ZC proxy	**0,65**	**0,84**	0,5	1,5	70,88%
Nwot	0,62	0,80	-	-	68,62%
LogSynflow	0,59	0,79	-	-	**71,34%**

Table 6.6: The table shows the Kendall-tau and Spearman correlation for Hybrid Zero-cost proxies on CIFAR100 dataset. Alpha and Beta here are the minimum values for which we obtain such results. Finally, the Top Accuracy is the accuracy value of the model found as optimal from that ZC proxy.

	Kendall-Tau	Spearman	Alpha	Beta	Top Accuracy
Hybrid ZC proxy	**0,64**	**0,82**	0,5	1,5	45,73%
Nwot	0,60	0,78	-	-	**45,9%**
LogSynflow	0,58	0,77	-	-	41,23%

Table 6.7: The table shows the Kendall-tau and Spearman correlation for Hybrid Zero-cost proxies on ImageNet dataset. Alpha and Beta here are the minimum values for which we obtain such results. Finally, the Top Accuracy is the accuracy value of the model found as optimal from that ZC proxy.

of 88,95% and 90,36%, respectively). While our experiments with the CIFAR10 dataset showed promising results for the Hybrid ZC proxy, it is important to note that the same level of performance was not observed in the other two datasets we evaluated. Specifically, the Hybrid ZC proxy was not able to find the highest accuracy models on CIFAR100 and ImageNet16-120, unlike its constituent proxies LogSynflow and Nwot. Further analysis revealed that the LogSynflow and Nwot ZC proxies were able to identify models with high accuracy rates on CIFAR100 and ImageNet16-120, respectively, which were competitive with the highest accuracy models on each dataset. For instance, on CIFAR100, where the model with the highest accuracy achieved a rate of 73,26%, the LogSynflow proxy identified a model with an accuracy rate of 71,34%. Similarly, on ImageNet16-120, where the model with the highest accuracy achieved a rate of 47,33%, the Nwot proxy found a model with an accuracy rate of 45,9%. It is worth noting that, in both cases described above, the Hybrid ZC proxy was able to identify models with only slightly lower accuracy rates.

Although our results have confirmed that the Zero-cost proxies and the new Hybrid ZC proxy can be useful in identifying accurate and efficient models, it is important to note that the current evaluation metrics, based on the Kendall-tau and Spearman correlation, may not be sufficient for assessing Zero-cost proxies.

Correlation measures evaluate the correspondence between the rankings of two lists, and a higher correlation may indicate a fairer ranking in the middle or at the end of the list of models, but it may not necessarily reflect the performance of the models ranked at the beginning of the list, where the best-performing models are located within the search space. Therefore, the ability of the ZC proxies to identify the best models more accurately cannot be reliably measured using correlation measures alone. To comprehensively evaluate the performance of zero-cost proxies, it is important to consider additional metrics beyond correlation measures, such as run time or memory usage, which can reflect the practicality and effectiveness of the identified models. Therefore, we recommend that future studies explore and adopt a comprehensive set of metrics to provide a more complete picture of the performance of zero-cost proxies in identifying accurate and efficient models. This will allow researchers to assess the practical usefulness of the identified models and better understand the trade-offs between model accuracy and efficiency.

Chapter 7

Conclusion

This thesis explores the rapidly-evolving field of Automated Machine Learning (AutoML) and Neural Architecture Search (NAS), two groundbreaking technologies that are driving a huge change in the field of machine learning. By conducting a thorough analysis and evaluation, this thesis brings attention to the significant potential of these tools also highlighting some of the critical challenges associated with Neural Architecture Search (NAS), particularly the considerable computational burden involved in finding and training the optimal neural network architectures. Furthermore, this thesis also explores the potential of Zero-cost (ZC) proxies in the context of AutoML and NAS, showing how they can be leveraged to reduce the computational costs and enhance the efficiency of the search for optimal machine learning models.

The Zero-cost proxies provide a way to estimate the performance of a candidate architecture without actually training and evaluating the model on the dataset. ZC proxies are usually simpler and faster than computing the actual performance metric of interest, and they can be used to quickly filter out candidate neural network architectures that are likely to perform poorly on the dataset.

The main contribution of this thesis is the creation of a new Hybrid Zero-cost proxy capable of outperforming current ZC proxies. The Hybrid Zero-cost proxy is obtained by combining the benefits of the two categories of ZC : Data-Independent (DI) and Data-Dependent (DD). Specifically, the Hybrid ZC proxy is created through a weighted sum of LogSynflow [1], which is the best performing ZC proxy DI, and Nwot [2] , which is the best performing ZC proxy DD. The performance of all model experiments was evaluated using three datasets: CIFAR10, CIFAR100, and ImageNet16-120, which are contained within the Nas-Bench-201 [22] benchmark. The results demonstrate a significant improvement in both the Kendall-Tau, about 7%, and Spearman, about 5 %, correlation values across all datasets compared to the most effective ZC proxies previously proposed, namely LogSynflow and Nwot. Moreover, the proposed Hybrid Zero-cost proxy has been demonstrated to possess the capability of identifying a superior performing model, with significantly higher accuracy, compared to the ones identified by the existing LogSynflow and Naswot ZC proxies.

This thesis presents several notable contributions, including the development of a novel Hybrid ZC proxy for identifying high-performing machine learning models. However, due to the constraints of time and computational resources, there are some limitations that should be addressed in future researches. Future studies could aim to increase the cross-sectional applicability of the Hybrid ZC proxy by conducting tests on different set of benchmarks and datasets. Additionally, incorporating the proposed ZC proxy into an AutoML pipeline could reveal its efficacy in more complex applications. Finally, to provide a rigorous foundation for the effectiveness of this measure, it would be valuable to develop a mathematical theory to support the Hybrid ZC proxy. This would allow for a deeper understanding of the underlying principles and help to optimize its implementation. Overall, the results and implications of this thesis are promising and offer significant potential for advancing the field of Automated Machine Learning and Neural Architecture Search.

Bibliography

[1] Niccolò Cavagnero, Luca Robbiano, Barbara Caputo, and Giuseppe Averta. «FreeREA: Training-Free Evolution-based Architecture Search». In: *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2023, Waikoloa, HI, USA, January 2-7, 2023*. IEEE, 2023, pp. 1493–1502. DOI: 10.1109/WACV56688.2023.00154. URL: https://doi.org/10.1109/WACV56688.2023.00154 (cit. on pp. i, 30, 41, 56).

[2] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. *Neural Architecture Search without Training*. 2020. DOI: 10.48550/ARXIV.2006.04647. URL: https://arxiv.org/abs/2006.04647 (cit. on pp. i, 28, 32, 41, 56).

[3] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. *Zero-Cost Proxies for Lightweight NAS*. 2021. DOI: 10.48550/ARXIV.2101.08134. URL: https://arxiv.org/abs/2101.08134 (cit. on pp. 2, 28, 30, 41, 42).

[4] *AutoML Lecture*. https://github.com/automl-edu/AutoMLLecture/blob/master/slides/w01_big_picture.pdf. Accessed: 2023-02-03 (cit. on pp. 9, 10, 12–14).

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. «Neural Architecture Search: A Survey». In: (2018). URL: https://arxiv.org/abs/1808.05377 (cit. on pp. 15, 16).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385 (cit. on p. 16).

[7] Barret Zoph and Quoc V. Le. *Neural Architecture Search with Reinforcement Learning*. 2016. DOI: 10.48550/ARXIV.1611.01578. URL: https://arxiv.org/abs/1611.01578 (cit. on pp. 17, 22, 25).

[8] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. *Learning Transferable Architectures for Scalable Image Recognition.* 2017. DOI: 10.48550/ARXIV.1707.07012. URL: https://arxiv.org/abs/1707.07012 (cit. on pp. 17, 36).

[9] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search.* 2018. DOI: 10.48550/ARXIV.1806.09055. URL: https://arxiv.org/abs/1806.09055 (cit. on pp. 17, 23, 24, 34, 36).

[10] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. *Hierarchical Representations for Efficient Architecture Search.* 2017. DOI: 10.48550/ARXIV.1711.00436. URL: https://arxiv.org/abs/1711.00436 (cit. on pp. 18, 21).

[11] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. «Efficient and Robust Automated Machine Learning». In: *Advances in Neural Information Processing Systems.* Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf (cit. on p. 20).

[12] James Bergstra and Yoshua Bengio. «Random search for hyper-parameter optimization.» In: *Journal of machine learning research* 13.2 (2012) (cit. on p. 20).

[13] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. *A Survey on Neural Architecture Search.* 2019. DOI: 10.48550/ARXIV.1905.01392. URL: https://arxiv.org/abs/1905.01392 (cit. on pp. 21, 22).

[14] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. *Progressive DARTS: Bridging the Optimization Gap for NAS in the Wild.* 2019. DOI: 10.48550/ARXIV.1912.10952. URL: https://arxiv.org/abs/1912.10952 (cit. on pp. 24, 26).

[15] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. *Efficient Neural Architecture Search via Parameter Sharing.* 2018. DOI: 10.48550/ARXIV.1802.03268. URL: https://arxiv.org/abs/1802.03268 (cit. on pp. 24, 25, 36).

[16] Chenxi Liu et al. *Progressive Neural Architecture Search.* 2017. DOI: 10.48550/ARXIV.1712.00559. URL: https://arxiv.org/abs/1712.00559 (cit. on p. 24).

[17] Stefan Falkner, Aaron Klein, and Frank Hutter. «BOHB: Robust and Efficient Hyperparameter Optimization at Scale». In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1437–1446. URL: `https://proceedings.mlr.press/v80/falkner18a.html` (cit. on p. 24).

[18] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. «Pruning neural networks without any data by iteratively conserving synaptic flow». In: (2020). DOI: `10.48550/ARXIV.2006.05467`. URL: `https://arxiv.org/abs/2006.05467` (cit. on pp. 28, 29, 41).

[19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. *SNIP: Single-shot Network Pruning based on Connection Sensitivity.* 2018. DOI: `10.48550/ARXIV.1810.02340`. URL: `https://arxiv.org/abs/1810.02340` (cit. on pp. 28, 29, 41).

[20] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. *Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective.* 2021. DOI: `10.48550/ARXIV.2102.11535`. URL: `https://arxiv.org/abs/2102.11535` (cit. on p. 28).

[21] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. *EcoNAS: Finding Proxies for Economical Neural Architecture Search.* 2020. DOI: `10.48550/ARXIV.2001.01233`. URL: `https://arxiv.org/abs/2001.01233` (cit. on p. 28).

[22] Xuanyi Dong and Yi Yang. *NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search.* 2020. DOI: `10.48550/ARXIV.2001.00326`. URL: `https://arxiv.org/abs/2001.00326` (cit. on pp. 28, 34–36, 56).

[23] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. *AtomNAS: Fine-Grained End-to-End Neural Architecture Search.* 2020. arXiv: `1912.09640 [cs.CV]` (cit. on p. 29).

[24] Chaoqi Wang, Guodong Zhang, and Roger Grosse. «Picking Winning Tickets Before Training by Preserving Gradient Flow». In: (2020). DOI: `10.48550/ARXIV.2002.07376`. URL: `https://arxiv.org/abs/2002.07376` (cit. on pp. 29, 41).

[25] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. «NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: `10.1109/tpami.2021.3054824`. URL: `https://doi.org/10.1109%2Ftpami.2021.3054824` (cit. on pp. 30, 31, 49).

[26] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. *Regularized Evolution for Image Classifier Architecture Search.* 2019. arXiv: 1802.01548 [cs.NE] (cit. on p. 30).

[27] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. «Nas-bench-101: Towards reproducible neural architecture search». In: *International Conference on Machine Learning.* PMLR. 2019, pp. 7105–7114 (cit. on pp. 31, 35).

[28] Arjun Krishnakumar, Colin White, Arber Zela, Renbo Tu, Mahmoud Safari, and Frank Hutter. *NAS-Bench-Suite-Zero: Accelerating Research on Zero Cost Proxies.* 2022. DOI: 10.48550/ARXIV.2210.03230. URL: https://arxiv.org/abs/2210.03230 (cit. on pp. 31, 32, 48, 50, 51).

[29] Michael Ruchte, Arber Zela, Julien Niklas Siems, Josif Grabocka, and Frank Hutter. «NASLib: a modular and flexible neural architecture search library». In: (2020) (cit. on pp. 31, 33).

[30] Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. «Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2021, pp. 5251–5260 (cit. on pp. 32, 35).

[31] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX.* Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008 (cit. on p. 34).

[32] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. «Nas-bench-301 and the case for surrogate benchmarks for neural architecture search». In: *arXiv preprint arXiv:2008.09777* (2020) (cit. on p. 35).

[33] Abhinav Mehrotra, Alberto Gil CP Ramos, Sourav Bhattacharya, Łukasz Dudziak, Ravichander Vipperla, Thomas Chau, Mohamed S Abdelfattah, Samin Ishtiaq, and Nicholas Donald Lane. «Nas-bench-asr: Reproducible neural architecture search for speech recognition». In: *International Conference on Learning Representations.* 2021 (cit. on p. 35).

[34] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. «Nas-bench-nlp: neural architecture search benchmark for natural language processing». In: *IEEE Access* 10 (2022), pp. 45736–45747 (cit. on p. 35).

[35] Alex Krizhevsky, Geoffrey Hinton, et al. «Learning multiple layers of features from tiny images». In: (2009) (cit. on p. 37).

[36] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. *A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets.* 2017. DOI: 10.48550/ARXIV.1707.08819. URL: https://arxiv.org/abs/1707.08819 (cit. on p. 37).

[37] Vasco Lopes, Saeid Alirezazadeh, and Luı s A. Alexandre. «EPE-NAS: Efficient Performance Estimation Without Training for Neural Architecture Search». In: *Lecture Notes in Computer Science.* Springer International Publishing, 2021, pp. 552–563. DOI: 10.1007/978-3-030-86383-8_44. URL: https://doi.org/10.1007%5C%2F978-3-030-86383-8_44 (cit. on p. 41).

[38] Jack Turner, Elliot J. Crowley, Michael O'Boyle, Amos Storkey, and Gavin Gray. *BlockSwap: Fisher-guided Block Substitution for Network Compression on a Budget.* 2019. DOI: 10.48550/ARXIV.1906.04113. URL: https://arxiv.org/abs/1906.04113 (cit. on p. 41).

[39] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. *Zen-NAS: A Zero-Shot NAS for High-Performance Deep Image Recognition.* 2021. DOI: 10.48550/ARXIV.2102.01063. URL: https://arxiv.org/abs/2102.01063 (cit. on p. 41).