UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Bruno Griep Fernandes

Efficient Trainingless Metric for Performance Evaluation in the Context of Neural Architecture Search

Florianópolis
2022

Bruno Griep Fernandes

Efficient Trainingless Metric for Performance Evaluation in the Context of Neural Architecture Search

Dissertação submetida ao Programa de Pós-Graduação
em Engenharia Elétrica da Universidade Federal de
Santa Catarina para a obtenção do título de Mestre
em Engenharia Elétrica.
Orientador: Prof. Eduardo Luiz Ortiz Batista, Dr.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Fernandes, Bruno Griep
   Efficient Trainingless Metric for Performance Evaluation in the Context of Neural Architecture Search / Bruno Griep Fernandes ; orientador, Eduardo Luiz Ortiz Batista, 2022.
   60 p.

   Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, 2022.

   Inclui referências.

   1. Engenharia Elétrica. 2. Aprendizagem de Máquina. 3. Pesquisa de Arquitetura Neural. 4. Sem Treinamento. I. Ortiz Batista, Eduardo Luiz . II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

Bruno Griep Fernandes

Efficient Trainingless Metric for Performance Evaluation in the Context of Neural Architecture Search

O presente trabalho em nível de Mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Richard Demo Souza, Dr.
Universidade Federal de Santa Catarina

Prof. Edson Cilos Vargas Júnior, Dr.
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Engenharia Elétrica.

ICP-Edu
Documento assinado digitalmente
Telles Brunelli Lazzarin
Data: 08/12/2022 08:37:20-0300
CPF: ***.400.719-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Telles Brunelli Lazzarin, Dr.
Coordenação do Programa de
Pós-Graduação em Engenharia Elétrica

ICP-Edu
Documento assinado digitalmente
Eduardo Luiz Ortiz Batista
Data: 06/12/2022 11:39:59-0300
CPF: ***.521.889-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Luiz Ortiz Batista, Dr.
Orientador

Florianópolis, 2022.

Dedico esta dissertação aos meus pais, Ruben e Vera, e
também à minha namorada Laís, pelo apoio e paciência
durante esta etapa de minha vida. Mas especialmente
dedico esta dissertação à minha falecida avó Glenda,
que nunca deixou de acreditar em meus estudos e sei
que estaria muito feliz com a conquista.

ACKNOWLEDGEMENTS

Desejo expressar meu reconhecimento a todos que, de uma maneira ou outra, colaboraram na realização deste trabalho.

Agradeço à minha família por todo o apoio, estabilidade e motivação durante minha graduação.

À minha namorada Laís, por toda a paciência e suporte nos momentos mais difíceis de realização desta dissertação.

Ao Eduardo Luiz Ortiz Batista, pelos anos de supervisão de aprendizados proporcionados, sejam eles como professor de graduação como orientador neste trabalho de dissertação de mestrado.

No matter how hard or how impossible it is, never lose sight of your goal. (Monkey D. Luffy in One Piece by Eiichiro Oda.)

ABSTRACT

Este trabalho é dedicado ao problema de Pesquisa de Arquitetura Neural (NAS) no contexto de redes neurais profundas. Em termos gerais, tal problema envolve algoritmos com longos períodos de execução que buscam a melhor arquitetura de rede neural possível para resolver um problema prático específico. Algumas abordagens sem treinamento foram propostas recentemente com o objetivo de resolver o problema do NAS em menores períodos de tempo de execução. Essas abordagens são baseadas em métricas que ajudam a prever o desempenho de arquiteturas de redes neurais sem treiná-las. Neste trabalho de dissertação, é apresentada uma nova métrica de desempenho para NAS sem treinamento. A métrica proposta é derivada da chamada métrica *NAS without Training* (NAS-WOT), visando simplicidade e maior velocidade de busca na rede. Outra contribuição deste trabalho é um esquema de penalização envolvendo a métrica proposta, que auxilia na busca por redes efetivas que também sejam pequenas em termos de complexidade computacional necessária para implementação de inferências. Resultados experimentais mostram a eficácia da métrica proposta e de sua versão penalizada.

Palavras-chave: Aprendizagem de Máquina. Pesquisa de Arquitetura Neural. Sem Treinamento.

RESUMO ESTENDIDO

INTRODUÇÃO

O uso de redes neurais profundas vem crescendo em muitos campos de pesquisas e para inúmeras aplicações. Com este crescimento o *design* de redes neurais se tornou um grande desafio para criar-se uma arquitetura neural que apresente melhorias em desempenho, acurácia, complexidade, etc. O *design* manual de redes neurais apresenta limitações claras, provenientes de metodologias envolvendo tentativa e erro ao testar e validar as redes propostas. Graças a essas limitações, um novo campo de pesquisa surgiu, sendo ele a automação de propostas de arquiteturas assim como sua validação e testes, campo este posteriormente chamado de Pesquisa de Arquitetura Neural, do inglês *Neural Architecture Search* (NAS). A Pesquisa de Arquitetura Neural é definida como a composição de três componentes: o espaço de busca, responsável pela definição do conjunto de operações e conexões da rede; a estratégia ou algoritmo de busca, responsável em definir como explorar o espaço pré-definido e como amostrar as arquiteturas candidatas dentre uma população; a estratégia de estimativa de desempenho, consistindo em como estimar, medir e predizer o desempenho de uma arquitetura sendo avaliada.

Este trabalho aborda o terceiro componente de um sistema de pesquisa de arquitetura neural, a estratégia de estimativa de desempenho, sendo no que lhe concerne o processo mais prolongado de um sistema de busca, pois a forma clássica de medida é treinar a rede proposta do zero, levando a horas de execução apenas para avaliar uma única rede dentre um grande conjunto.

OBJETIVOS

O objetivo deste trabalho é desenvolver uma métrica sem treinamento para auxiliar a etapa de estimativa de desempenho de um sistema de pesquisa de arquitetura neural, com foco na redução do tempo de busca para as já rápidas abordagens sem treinamento.

METODOLOGIA

Inicialmente é apresentada uma revisão sobre *Neural Architecture Search*, que inclui sua definição, princípios, desafios e possibilidades. Para cada módulo de NAS é apresentada sucintamente diferentes abordagens e metodologias. A revisão é finalizada com as abordagens sem treinamento para um dos estágios que compõem um sistema NAS.

Posteriormente, são apresentadas as contribuições deste trabalho. Inicialmente é apresentada a examinação de uma métrica existente na literatura, descrevendo-a por completo, desde a sua formulação em forma matricial, sua definição se é ou não uma matriz positiva semi-definida, o papel de seus autovalores e por fim seus aspectos práticos.

Em seguida é apresentada a métrica proposta neste trabalho, com o intuito de reduzir a complexidade computacional e reduzir o tempo de cálculo em contraste a métrica já existente na literatura. É demonstrada a forma com o qual calcula-se a métrica proposta e discute-se o motivo pelo qual a mesma pode comportar-se como um preditor de complexidade computacional.

RESULTADOS E DISCUSSÕES

O método proposto é avaliado em quatro experimentos diferentes, dois destes comparando o modelo proposto com outra metodologia sem treinamento. O primeiro experimento trata-se de amostrar 1000 arquiteturas de redes dentre um dos possíveis *benchmarks* e conjuntos de dados selecionados, cada uma destas arquiteturas é avaliada pela métrica proposta e pela métrica encontrada na literatura. Posteriormente as métricas são avaliadas utilizando o coeficiente de correlação de Kendall em relação à acurácia final de cada arquitetura.

O segundo experimento tem como intenção avaliar o tempo de execução para três cenários, 10, 100 ou 1000 redes aleatoriamente amostradas, gerando como resultado uma tabela com os tempos de execução, a melhor acurácia encontrada utilizando a métrica proposta e a disponível na literatura.

Os dois últimos experimentos são a reprodução dos dois primeiros, porém utilizando o tamanho em kilobytes da rede como fator de penalização. Considerando isto, algumas funções foram definidas para aplicar esta penalização. A métrica proposta demonstra grande correlação com o tamanho da rede, o que torna-a melhor para avaliar a complexidade computacional de uma arquitetura neural. O quarto experimento demonstra quanto tempo a mais seria necessário para aplicar a penalização e o resultado, sendo possível encontrar redes com poucos kilobytes, porém com acurácias elevadas.

CONCLUSÕES

Esta dissertação propõe uma nova métrica livre de treinamento para a etapa de estimativa de desempenho de um sistema NAS, que pontua a rede não treinada na inicialização, com uma correlação positiva em relação a sua acurácia final após treinamento, além de possuir uma correlação positiva com o tamanho da rede também. Os resultados demonstram que o método proposto está alinhado com o objetivo principal do trabalho, que é de filtrar arquiteturas com baixo desempenho.

Keywords: Machine Learning. Neural Architecture Search. Training-less.

ABSTRACT

This work is dedicated to the problem of network architecture search (NAS) in the context of deep neural networks. In general terms, such a problem involves algorithms with long running times that look for the best possible neural network architecture for solving a specific practical problem. Some trainingless (or training-free) approaches have been proposed recently aiming to tackle the NAS problem in shorter times. These approaches are based in performance metrics that help predicting the performance of neural network architectures without training them. In this dissertation work, a novel performance metric for trainingless NAS is presented. The proposed metric is derived from the so-called NAS without training (NAS-WOT) metric, aiming at simplicity and further network-search speed ups. Another contribution of this work is a penalization scheme involving the proposed metric, which helps searching for effective networks that are also small in terms of computational complexity required for inference implementation. Experimental results show the effectiveness of the proposed metric and of its penalized version.

Keywords: Machine Learning. Neural Architecture Search. Training-less Neural Architecture Search.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The use of deep neural networks has been growing in many fields and applications, both for industry and academic purposes. The design of high-performance effective neural networks is a challenging task, particularly due to the vast and complex range of possibilities involving the different parts of a neural network. Manually-designed neural networks often present limitations, which is due to the trial-an-error methodology normally adopted in their design. Such a methodology together with the need to provide solutions in a fixed time span, tend to result in networks that meet minimal requirements, but that do not go further on an extensive search for a better solution. Motivated by such a problem, a great research effort has been dedicated to develop systems that can automatically search and design neural networks architectures. This effort has given rise to the so-called neural architecture search (NAS) methods.

As a sub-field of automated machine learning (AutoML) [1], NAS aims at automating the architecture design of a neural network. To meet this aim, different techniques has been used, such as reinforcement learning [2] and evolutionary algorithms [3]. Regardless of the core technique used, the majority of works in the field describes NAS as a composition of three major components: the search space, the search strategy, and the performance estimation strategy [4]. The search space defines a set of operations (e.g., convolution, full connection, pooling, among others) and how these operations can be connected to form a valid network architecture. In other words, the search space defines which architectures can be represented. The search strategy (or algorithm) is the name given to the module that defines how to explore the search space or how to effectively sample a population of network architecture candidates. This module receives the child model performance metrics, involving an exploration-exploitation trade-off, and optimizes to generate high-performance candidates. The last part of NAS, and the focus of this work, is the performance estimation strategy. This part consists of estimating, measuring, and/or predicting the performance of a candidate architecture provided by the search strategy.

NAS systems are broadly based on the works in [2] and [5], which achieved state-of-the-art architectures for image recognition and language modeling, respectively. In these works, a controller is used to generate architecture proposals derived from the basic set of operations in a pre-defined search space. The networks are then trained to provide a “reward” for updating the controller, which can then propose a new supposedly better architecture and repeat the process. One of the main problems with such an approach is that training a candidate architecture for every controller update results in a high computational cost. For instance, 800 GPUs (Graphics Processing Units) were used for 28 days to achieve the results in [2]. This large amounts of time spent running a standard NAS system motivated the improvements introduced in subsequent works

([6], [7]) but, despite such improvements, standard NAS still tends to be slow for many real-world applications. This motivated the development of trainingless (or training-free) NAS methods [8] [9] [10], in which the performance of a candidate is estimated using some particular performance metric, avoiding the cost of training the candidate network.

1.1 OBJECTIVE

The present work has the main objective of developing an efficient training-free metric for performance estimation in NAS systems. The focus is on reducing the search time for the already fast trainingless NAS approaches.

1.2 DOCUMENT STRUCTURE

The remainder of this document is structured as described in the following. Chapter 2 gives an overview on neural architecture search (NAS), describing the evolution and the different components of a NAS system in detail. Chapter 3 describes the contributions of this work, namely a detailed analysis of the well-known NAS without training (NAS-WOT) metric, the proposed metric, and a discussion regarding a modification of the proposed metric aiming to facilitate the search for effective networks with low computational complexity. In Chapter 4, experimental results are described aiming to assess the effectiveness of the proposed metric as well as of its aforementioned modification. Finally, in Chapter 5, the conclusions of this work are presented.

2 BACKGROUND ON NEURAL ARCHITECTURE SEARCH

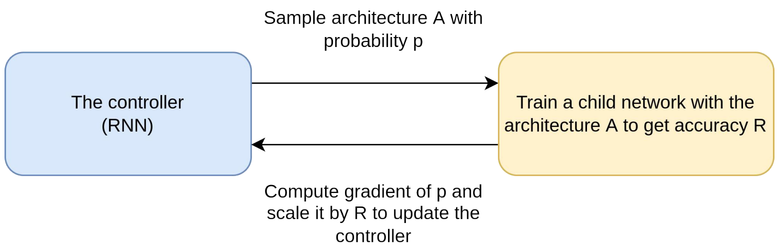
The work by Zoph and Le [2] was plausibly the kickoff proposal that made NAS methods more broadly known. There, the authors use an approach based on recurrent neural networks (RNN) to generate sample network architectures, train such architectures, and record the obtained accuracy. A gradient-based method is then used for updating the controller aiming to improve its search for the best architecture. Such a NAS solution exemplifies the rationale behind the earlier NAS solutions [2], [3], [5], which basically work in a loop process comprising the following steps:

1. The controller generates an architecture;
2. The generated architecture is trained and evaluated;
3. The obtained result is used as a reward to the controller to update its parameters accordingly;
4. Previous steps are repeated *N* times.

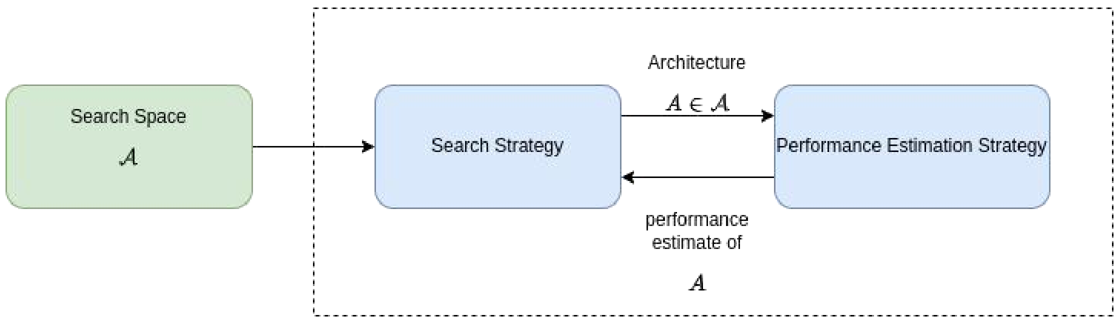
In this context, the controller is basically implementing a **search strategy** that tries to effectively sample an architecture *A* from a **search space** of multiple possible architectures $\mathcal{A}$ according to some **performance metric**. In the following sections, these three important aspects of NAS are discussed, namely the search space, search strategy and performance metric. This process is illustrated in Fig. 1.

Figure 1 – General neural architecture search flow.

(a) General Neural Architecture Search prototype in the form of a loop.


Sample architecture A with probability p
The controller (RNN)
Train a child network with the architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller

(b) Abstract illustration with the three main components of Neural Architecture Search method. A search strategy selects an architecture *A* from a predefined search space $\mathcal{A}$. An evaluation of the architecture is executed by the performance estimation strategy, which return the estimated performance of *A* to the search strategy.



Source: Adapted from [2].

2.1 SEARCH SPACE

The search space is basically defined by the parameters that characterize a network architecture, such as the number of layers, the type of each layer, how they are connected, among others [2], [3], [5]. This section explores some of the methodologies used to define search spaces in the context of NAS.

2.1.1 Global Search Spaces

A relatively simple way to design a search space for neural network architectures is by considering architectures with sequential layer-wise operations. The resulting chain-structured neural networks consists of a sequence of N layers, with the i–th layer L_i receiving its input from layer L_{i-1}. The network architecture is then parameterized by the number of layers N; the type of each layer (e.g., convolutional, pooling, fully connected, among others); and hyperparameters associated with each layer. In Fig. 2(a), an example of a direct chain structure such as the one described is shown. On the other hand, Fig. 2(b) illustrates a more complex chain structure having layers operating in parallel as well as skip connections between the layers.

The global search space is defined as the space of chain-structured network architectures obtained considering all variations of the parameters described above. Such a search space, which is part of the early NAS proposals [5] [2], has a lot of potential regarding representation capability. However, this capability comes at the cost of a requirement of huge computational resources to cover all possibilities. To illustrate this point, In Zoph's and Le's experiments [2], 800 graphics processing units (GPUs) running in parallel for 28 days were necessary to cover all the search space considered.

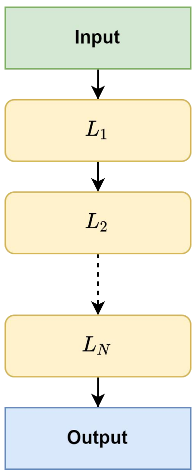
2.1.2 Cell-based Search Spaces

In [6], Zoph at al. introduced a new type of search space called NASNet (Neural Architecture Search Network), which was inspired by successful architectures like ResNets (Residual Neural Networks). This new search space makes use of repeated modules (or cells) containing a group of operations, which implies changing the granularity of the search space. The cells in the NASNet search space can be classified as: *normal cell*, which preserves the dimensionality of the input, and *reduction cell*, in which the output feature map has a reduced dimension. The final network architecture is obtained by stacking normal and reduction cells in a predefined way, resulting in a combination that is illustrated in Fig. 3.

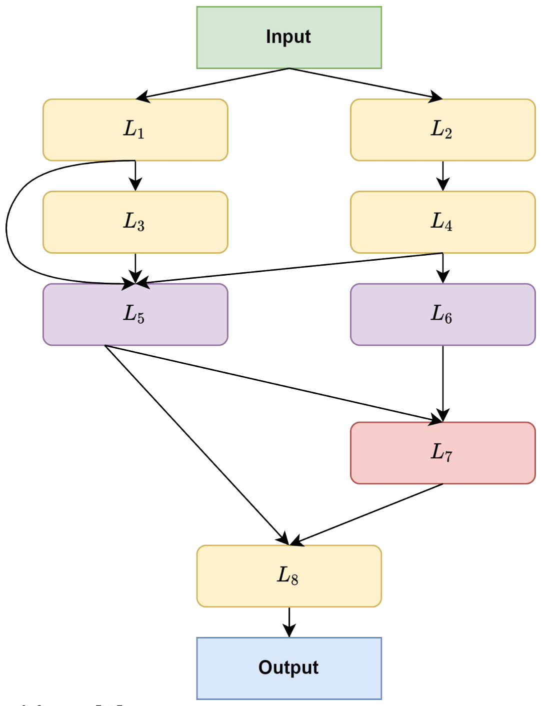
The cell-based search space has three major advantages compared by the previous works with sequential layer-wise operations. The first one is that the size of the search space is reduced drastically. In [6], the authors computed a 7x speed-up compared to their previous work [2], while achieving better performance and becoming the

Figure 2 – Chain-structured neural network without and with skip connections.

(a) Simplest example of a *chain-structured neural networks*.

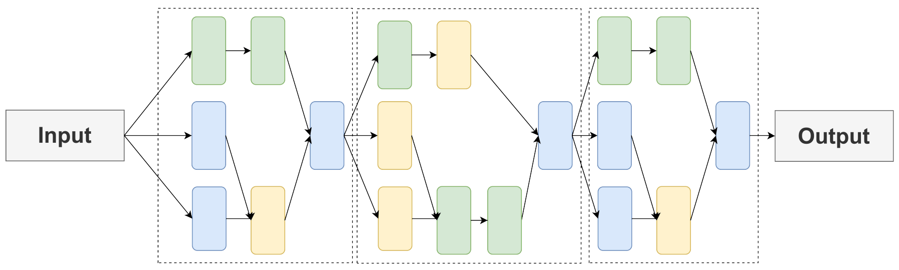


(b) Complex *multi-branch networks*.



Source: Adapted from [4].

Figure 3 – Complex chain-structured neural network with skip connections.



Source: Adapted from [4].

state-of-the-art for image recognition by the time. The second advantage is transferability of architectures built around cells, making the architecture more easily adaptable between different datasets. In [6], cells optimized to CIFAR-10 were transferred to the ImageNet case, achieving state-of-the-art performance. The third advantage is that creating architectures by repeating building blocks is a strong proof of a useful design pattern. However, even with all the aforementioned advantages, the algorithm required a total of 4 days running 500 GPUs in parallel, which corresponds to 2000 GPU days, to cover all the search space considered in [6].

2.1.3 Hierarchical Search Space

The hierarchical search space, introduced in [11], involves different levels of abstraction that allows overcoming some disadvantages of cell-based approaches. This representation has a balanced efficiency for an expressive search space, being the strongest advantage of eliminating the need to implement Bayesian, Evolutionary, Reinforcement Learning, and others complex search algorithms. The Hierarchical Neural Architecture Search (HNAS) approach, which uses such a search space, is the basis for the NAS strategy [6], which aims at finding architectures that can fit to a resource constrained environment, in this case the MCUs (Microcontroller Units).

2.2 SEARCH STRATEGIES

In Section 2.1, some methodologies used to define search spaces were described, showing how these spaces are designed but not how they are explored. The search strategy (or algorithm), which is another key component in NAS systems, is responsible to determine how to explore a search space, helping to identify the best performing architectures while avoiding the bad ones. The present section discusses some of the search strategies used for NAS.

2.2.1 Grid and Random Search

The grid and random search strategies are likely the simplest types of search strategies. The grid search is the name given when the search space is screened by sampling one candidate after other in a sequential way. The random search, in its turn, involves picking candidates randomly at each iteration (without replacement). Both strategies are viable for small search spaces, with the random search proving to be quite useful in the domain of hyperparameter search [12] (outperforming grid search when the number of parameters are not small). The disadvantage of grid and random searches is that both fail (i.e., become very slow) as the search space is increased.

2.2.2 Search Based on Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning (ML) where, instead of mapping inputs to a target, the algorithm attempt to map inputs to optimal actions. An agent interacts with its environment by selecting actions, in an effort to maximize its reward. The agent may be rewarded after each action, or only at specific points. By repeatedly interacting with the environment and observing its rewards, the agent improves its ability to select highly rewarding actions or series of actions [13].

The initial NAS design proposed by Zoph and Le [2] make use of an RL-based controller to guide the search process. Basically, the controller is implemented as a

RNN that sequentially generates architectures. Controller training is carried out using the REINFORCE algorithm [13]. The follow-up work from Zoph et al. [6] also uses and RL-based controller, but now using proximal policy optimization [14] to train the controller. In both works, the controller's action space is dictated by the search space. In that way the controller task have his *action space* as a list of tokens for defining a child network predicted by the controller. The reward is the accuracy achieved after a child network training process.

2.2.3 Search Based on Evolutionary Algorithms

Generally speaking, evolutionary algorithms utilize a population of individuals (mimicking real organisms, with the definition of genes and genomes) to create new individuals (an offspring) of increasingly better performance [15]. Each gene contains a piece of information about the problem being optimized. Genes can mutate to change their information and individuals can reproduce by crossover. In the context of the application of evolutionary algorithsm to NAS, individuals are network architectures and the mutations are carried out changing local operations (e.g, adding and removing layers, changing hyperparameters, skipping connections, among others). After training the offspring, their performance are evaluated, and they are added to the population. This procedure is illustrated in Fig. 4.

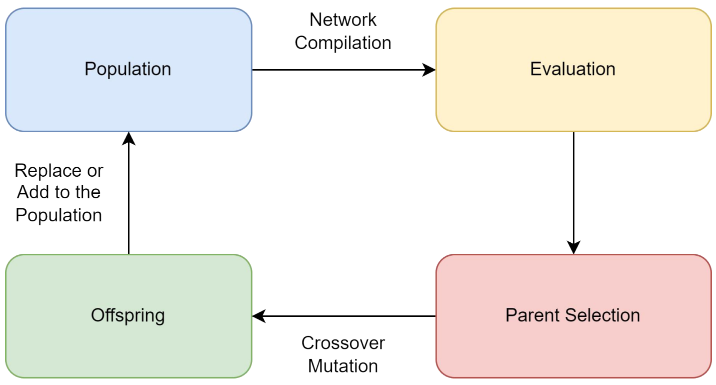
Evolutionary methods applied to neural networks differs in how they sample parents, update populations, and generate offspring. For example, some approaches [3], [11] use tournament selection [16] to sample parents. This method picks the best candidate out of a random set of samples, at each iteration, and places its mutated offspring into the population. When the tournament size is equal to one, tournament selection is the equivalent to a random selection. Another selection method, called *aging evolution* (AE), is used by the AmoebaNet algorithm [11]. In this method, tournament selection is modified aiming to prioritize younger genotypes and, thus, older individuals are discarded at each cycle. This allows AmoebaNet to cover and explore a larger part of the search space, avoiding to narrow down on good performance individuals too early in the process.

2.2.4 Bayesian Optimization

Bayesian optimization (BO) [18] is the leading technique for standard hyperparameter optimization processes. In such processes, an extensive search is carried out aiming to find hyperparameters that lead to the best validation accuracy. Thus, one can notice that there is a significant similarity between hyperparameter optimization and NAS.

One of the first apporches that use BO in the context of NAS can be found in [19]. There, the authors derived kernel functions for NAS using classic gaussian

Figure 4 – An example of the procedure at each evolutionary algorithm.



Source: Adapted from [17].

process-based Bayesian optimization (GPBO) methods, and then employ evolutionary algorithms to optimize the acquisition function to evaluate the neural architectures on the original objective function. Another work uses sequential model-based optimization (SMBO) as search strategy, resulting is a system called *Progressive NAS* (PNAS) [11]. In this work, the authors searched for architectures of increasing complexity (increasing the number of layer blocks in a cell) while utilizing a model to predict the accuracy of the obtained network.

2.3 PERFORMANCE EVALUATION STRATEGY

Another important aspect of NAS is performance evaluation or, more specifically, the strategy used for evaluating the performance of a new candidate architecture sampled from the search space. This aspect is crucial for a faster exploration of the search space and for obtaining an effective architecture in a reasonable amount of time. This section describes some of the different performance evaluation strategies used in the context of NAS.

2.3.1 Training from Scratch

Training from scratch is the most naive approach for performance evaluation. The idea is literally to train every new candidate or child network from scratch. This strategy, used originally in [2], results in extremely high computational costs, normally leading to thousands of GPU hours to finish a NAS task.

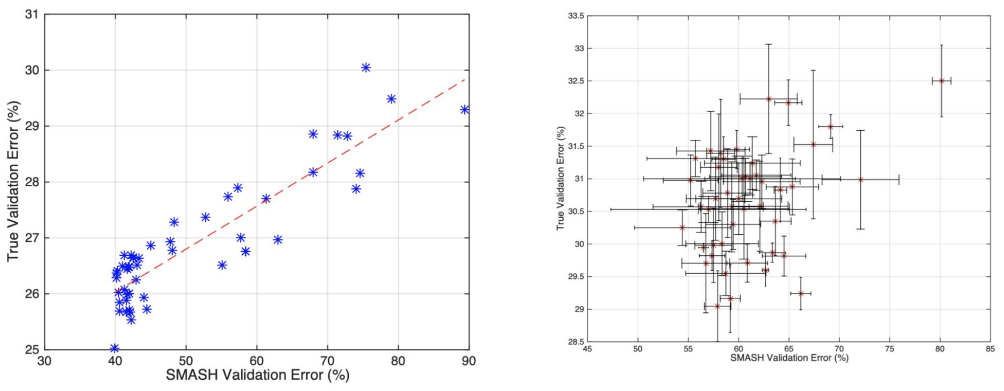
2.3.2 Proxy Task Performance Evaluation

Proxy task performance evaluation consists of using a smaller dataset, called proxy dataset, to evaluate the performance of the candidate/child network. In doing so and training the network for fewer epochs, the training time is reduced significantly, as shown in [6]. In spite of such a reduction, this approach tends to produce a biased performance estimate.

2.3.3 Prediction-Based Performance Evaluation

Another approach to speed-up performance assessment for candidate/child network architectures is introduced in [20]. Such an approach is based on using a HyperNet [21] to generate weights for a child network based on its architecture. The model with HyperNet-generated weights is validated directly, dispensing extra training for every child model, with the HyperNet training as a trade-off. A system called SMASH [20] demonstrate the potential of such an approach. However, as illustrated in Fig. 5, if the HyperNet model is relatively small as compared to the child model, the performance estimate obtained using the predicted weights is no longer useful for performance comparison, since the correlation between performance estimate and true validation errors is broken. Furthermore, recent works pointed out another problem of the SMASH system [7], which is the fact that the usage of HyperNet restricts the weights of the proposed child models to a low-rank space.

Figure 5 – SMASH algorithm results compared with true validation errors.



(a) A correlation between true error vs SMASH predicted error

(b) The correlation is broken when the HyperNet is too small.

Source: [20].

2.4 μNAS: A NAS SUCCESS CASE

The μNAS system [22] is a NAS application case that illustrates the potential as well as the different aspects (search space, search strategy and performance evaluation) of NAS strategies. The goal of the μNAS is to find small-yet-powerful microcontroller-level network architectures. To this end, the three scarce microcontroller resources are targeted, namely random access memory (RAM), persistent storage (flash), and processing speed. To handle these limitations, the authors use a highly-granular search space; a set of constraints that accurately capture resource scarcity of MCU platforms; a search algorithm capable of optimizing for multiple objectives in the search space; and network pruning, to obtain a small and accurate architecture. The authors also considered that an MCU search space would involve small architectures with restrictions on layer connectivity. The degrees of freedom considered for defining the search space are described in Table 1, which lead to a search space that comprises 1.15×10^{152} different models. The proposed search algorithm, based on aging evolution (AE) and Bayesian optimization (BO), navigates the search space by generating random architectures and applying morphisms to produce derived child networks [22].

Table 1 – Template for candidate models with free variables, their morphisms and bounds, defining the μNAS highly-granular search space.

Degree of freedom	**Options**	**Morphisms**
An architecture is N "convolutional" blocks, where each:	N in [1; 10]	append or remove random
• connects either in series or in parallel to the previous one	{*parallel, serial*}	change one to other
• has M convolution layers, where each layer:	M in [1; 3]	insert or remove random
• optionally, has a preceding 2×2 max-pooling operation;	{*yes, no*}	change one to other
• is either a full or a depthwise convolution with stride S, C	{*full, d/wise*}	change types
channels $K \times K$ kernel size (S = 1 for 1×1 convolution and C is	K in {1, 3, 5, 7}	change K by ± 2
not configured for depthwise convolution)	C in [1; 128]	change C by ± 1, 3, 5
	S in {1, 2}	change S by ± 1
• is optionally followed by batch norm.;	{*yes, no*}	change one to other
• is optionally followed by ReLU;	{*yes, no*}	change one to other
followed by a $P \times P$ average or maximum pooling,	{*avg, max*}	change one to other
	P in {2, 4, 6}	change P by ± 2
followed by F fully-connected layers, where each:	F in [1; 3]	insert or remove random
• has U units (output dimension), followed by a ReLU	U in [10; 256]	change U by ± 1, 3, 5
followed by a final fully-connected layer (U = number of classes).		

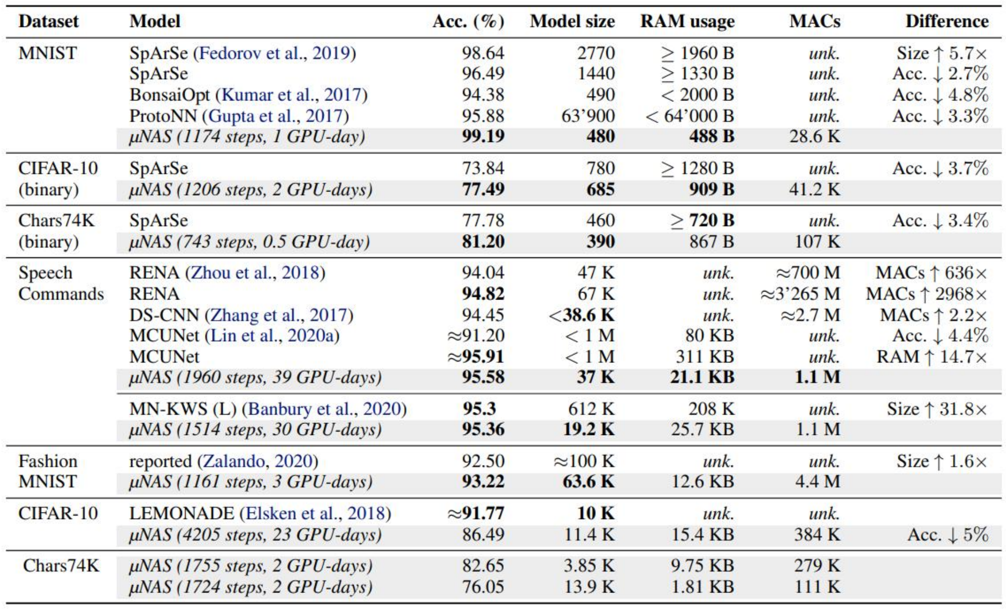
Source: [22].

The results presented by the authors, reproduced in Table 2, are quite impressive, showing that through a good design of the search space and explicit targeting of the primary resource bottlenecks, a NAS system can discover resource-efficient architectures. Nevertheless, to achieve these results for larger datasets, such as Speech Commands [23], μNAS requires a run time of 39 GPU-days for 1960 steps. The long run time is mostly caused by the training of each architecture proposed by the search algorithm, which consists of the biggest disadvantage of the μNAS system.

2.5 TRAININGLESS APPROACHES

As described in the previous sections, NAS methods can be quite effective in finding very good neural-network architectures conditioned to a good choice of search space and search algorithm. However, obtaining such architectures is a costly process that takes several hours or even days to finish. Examples of such costs are abundant: the approach in [2] required 800 GPUs running for 28 days to reach their results;

Table 2 – Comparison between μNAS results and known baselines.



Dataset	Model	Acc. (%)	Model size	RAM usage	MACs	Difference
MNIST	SpArSe (Fedorov et al., 2019)	98.64	2770	$\geq$ 1960 B	*unk.*	Size ↑ 5.7×
	SpArSe	96.49	1440	$\geq$ 1330 B	*unk.*	Acc. ↓ 2.7%
	BonsaiOpt (Kumar et al., 2017)	94.38	490	< 2000 B	*unk.*	Acc. ↓ 4.8%
	ProtoNN (Gupta et al., 2017)	95.88	63'900	< 64'000 B	*unk.*	Acc. ↓ 3.3%
	μNAS (1174 steps, 1 GPU-day)	**99.19**	**480**	**488 B**	28.6 K	
CIFAR-10 (binary)	SpArSe	73.84	780	$\geq$ 1280 B	*unk.*	Acc. ↓ 3.7%
	μNAS (1206 steps, 2 GPU-days)	**77.49**	**685**	**909 B**	41.2 K	
Chars74K (binary)	SpArSe	77.78	460	**$\geq$ 720 B**	*unk.*	Acc. ↓ 3.4%
	μNAS (743 steps, 0.5 GPU-day)	**81.20**	**390**	867 B	107 K	
Speech Commands	RENA (Zhou et al., 2018)	94.04	47 K	*unk.*	≈700 M	MACs ↑ 636×
	RENA	**94.82**	67 K	*unk.*	≈3'265 M	MACs ↑ 2968×
	DS-CNN (Zhang et al., 2017)	94.45	**<38.6 K**	*unk.*	≈2.7 M	MACs ↑ 2.2×
	MCUNet (Lin et al., 2020a)	≈91.20	< 1 M	80 KB	*unk.*	Acc. ↓ 4.4%
	MCUNet	≈**95.91**	< 1 M	311 KB	*unk.*	RAM ↑ 14.7×
	μNAS (1960 steps, 39 GPU-days)	**95.58**	**37 K**	**21.1 KB**	**1.1 M**	
	MN-KWS (L) (Banbury et al., 2020)	**95.3**	612 K	208 K	*unk.*	Size ↑ 31.8×
	μNAS (1514 steps, 30 GPU-days)	**95.36**	**19.2 K**	25.7 KB	1.1 M	
Fashion MNIST	reported (Zalando, 2020)	92.50	≈100 K	*unk.*	*unk.*	Size ↑ 1.6×
	μNAS (1161 steps, 3 GPU-days)	**93.22**	**63.6 K**	12.6 KB	4.4 M	
CIFAR-10	LEMONADE (Elsken et al., 2018)	≈**91.77**	**10 K**	*unk.*	*unk.*	
	μNAS (4205 steps, 23 GPU-days)	86.49	11.4 K	15.4 KB	384 K	Acc. ↓ 5%
Chars74K	*μNAS (1755 steps, 2 GPU-days)*	82.65	3.85 K	9.75 KB	279 K	
	μNAS (1724 steps, 2 GPU-days)	76.05	13.9 K	1.81 KB	111 K	

Source: [22].

MetaQNN [5] spent 96 GPU-days; and the aforementioned μNAS [22] required up to 39 GPU-days. All these costs motivated the development of the trainingless NAS approaches, in which performance evaluation is carried out without network training. These approaches are discussed in this section.

2.5.1 NAS-WOT

To the best of our knowledge, the first trainingless NAS approach is the so-called NAS without training (NAS-WOT) [9]. Such an approach is based on a particular performance metric that is simple to calculate and, in spite of not being a perfect performance metric for network architectures, it produces a rough estimate that at least allows discarding bad architectures.

The NAS-WOT performance metric is based on observing the activation patterns of the ReLU activation functions in a neural network. The idea is to expose the network to a randomly-selected mini-batch of input data $X = \{\mathbf{x}_k\}_{k=1}^{K}$ at initialization and, for each input sample $\mathbf{x}_k$, obtain a binary activation vector $\mathbf{c}_k$. The elements of $\mathbf{c}_k$ corresponding to non-activated ReLU units are equal to 0, whereas the elements corresponding to activated ReLU units are equal to 1. As described in [9], the intuition to the NAS-WOT metric is that the more similar the binary activation codes associated with two inputs are, the more challenging it is for the network to learn how to separate the corresponding samples. In other words, similar ReLU activation patterns for different input samples indicate that the network will have more difficulty separating these samples.

Aiming to quantify the difference between activation vectors and considering that

$\mathbf{c}_k$ $\forall k$ are binary vectors, the Hamming distance is used. The Hamming distance can be defined as:

$$d_H(\mathbf{u}, \mathbf{v}) = |\{i; u_i v_i, i \leq i \leq n\}|, \tag{1}$$

resulting in the information of the number of places in which $\mathbf{u}$ and $\mathbf{v}$ differ. Thus, one has $d_H(\mathbf{c}_i, \mathbf{c}_j)$ to represent the Hamming distance between $\mathbf{c}_i$ and $\mathbf{c}_j$. Then, the following kernel matrix is constructed:

$$\mathbf{K}_\mathrm{H} = \begin{bmatrix} N_\mathrm{A} - d_H(\mathbf{c}_1, \mathbf{c}_1) & \dots & N_\mathrm{A} - d_H(\mathbf{c}_1, \mathbf{c}_K) \\ \vdots & \ddots & \vdots \\ N_\mathrm{A} - d_H(\mathbf{c}_K, \mathbf{c}_1) & \dots & N_\mathrm{A} - d_H(\mathbf{c}_K, \mathbf{c}_K) \end{bmatrix} \tag{2}$$

where N_A is the number of ReLU activations in the network. The elements at the main diagonal of $\mathbf{K}_\mathrm{H}$ will always be equal to N_A, since $d_H(\mathbf{c}_i, \mathbf{c}_i) = 0, \forall i$. Moreover, elements outside the main diagonal (e.g., at a position i,j) will be closer to N_A if the corresponding codes $\mathbf{c}_\mathrm{i}$ and $\mathbf{c}_\mathrm{j}$ are similar, or will become small as the difference between $\mathbf{c}_\mathrm{i}$ and $\mathbf{c}_\mathrm{j}$ grows.

Finally, the NAS-WOT performance metric s is calculated according to

$$s = \log |\mathbf{K}_\mathrm{H}| \tag{3}$$

where $|\mathbf{K}_\mathrm{H}|$ represents the determinant of $\mathbf{K}_\mathrm{H}$, the logarithm operator was used to avoid overflows during the computational process. As described in [9], if $\mathbf{K}_\mathrm{H}$ is closer to an identity matrix (i.e., all binary activation vectors $\mathbf{c}_i$ $\forall i$ are very different), s will result in higher values. On the other hand, if all binary activation vectors are similar, $|\mathbf{K}_\mathrm{H}| = 0$, then s will tend to $-\infty$. This metric encodes the previously described intuition behind the NAS-WOT approach.

The experiments conducted by the authors were performed using the NATS-Bench TSS [24], NATS-Bench SSS [25], NASBench101 [26], and NDS Facebook [27] benchmarks. Considering the experiments results, a positive correlation between the validation accuracy and the NAS-WOT performance metric for all the benchmarks, in addition it was required only a minute to evaluate more or less than 100 networks.

The major benefit of NAS-WOT performance metric is its rapid execution time. The metric can be explore and used to speedup performing architecture search stage, specially as a fast filtering step, eliminating the low accuracy networks without being necessary the training step to evaluate the bad networks.

2.5.2 EPE-NAS

Another trainingless NAS approach, termed EPE-NAS (Efficient Performance Estimation without Training for Neural Architecture Search), was introduced in [9]. The starting point for EPE-NAS is the fact that, for a given data point (i.e., a given input vector), a ReLU-based network behaves as a linear map between input and output. The

Table 3 – Comparison between NAS without Training [9] and EPE-NAS [8] evaluated using NAS-Bench-201 benchmark [24]. Performance shown in accuracy with mean$\pm$std, on CIFAR-10, CIFAR-100 and ImageNet-16-120.

Method	**Search (s)**	**CIFAR-10**		**CIFAR-100**		**ImageNet16-120**	
		validation	**test**	**validation**	**test**	**validation**	**test**
Training-free							
NAS-WOT (N=10)	3.1	89.56 ± 0.33	92.47 ± 0.04	69.34 ± 1.05	57.65 ± 30.61	42.08 ± 1.61	**42.20 ± 1.37**
EPE-NAS(N=10)	**2.3**	89.90 ± 0.21	**92.63 ± 0.32**	69.78 ± 2.44	**70.10 ± 1.71**	41.73 ± 3.60	41.92 ± 4.25
NAS-WOT (N=100)	25.7	89.91 ± 0.80	91.41 ± 2.24	67.13 ± 4.03	67.18 ± 4.14	41.39 ± 1.13	41.42 ± 1.53
EPE-NAS (N=100)	**20.5**	88.74 ± 3.16	**91.59 ± 0.87**	67.28 ± 3.68	**67.19 ± 3.82**	38.66 ± 4.75	**38.80 ± 5.41**
NAS-WOT (N=1000)	252.6	89.60 ± 0.90	91.20 ± 2.04	68.57 ± 0.41	68.95 ± 0.72	38.01 ± 1.66	38.08 ± 1.58
EPE-NAS(N=1000)	**206.2**	87.87 ± 0.85	**91.31 ± 1.69**	69.44 ± 0.83	**69.58 ± 0.83**	41.86 ± 2.33	**41.84 ± 2.06**

Source: [8].

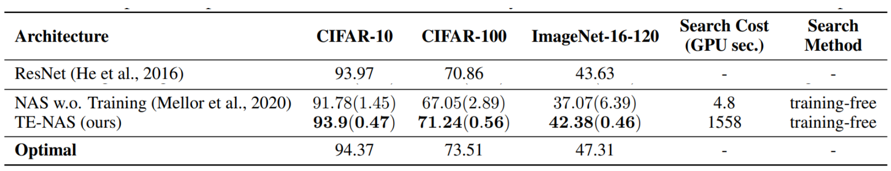
parameters of such a map can thus be obtained by taking the gradient of the output with respect to the input vector. The mappings for different data points of the same class can then be grouped and their similarity can be evaluated via the construction of a correlation matrix. The rationale, at this point, is that similar mappings are desirable for similar data points (i.e., data points belonging to the same class). Finally, the different correlation matrices for the different classes are considered for obtaining the EPE-NAS score. Normalization is also used to account for differences in the number of data points for each class.

From the results presented in [9], which are partially reproduced in Table 3, one can notice that the EPE-NAS outperforms the NAS-WOT approach in terms of both attained performance and speed. More specifically regarding the speed aspect, EPE-NAS reduced the search time by 16.5% to 25.8%.

2.5.3 TE-NAS

Another trainingless NAS approach, called TE-NAS (*training-free neural architecture search*), is presented in [28]. Such an approach is based on a new training-less metric composed of two indicators, which are based on recent advances in deep learning theory. Along with these two indicators, the authors introduced a pruning-based mechanism, to boost search efficiency and to obtain better performance. Fig. 6 reproduces the comparison between TE-NAS and NAS without Training [9] reported in [28]. From these results, one observes that the TE-NAS results in improvements in terms of accuracy of the obtained architecture, but at the cost of a search time that is 300 times higher than that of the NAS-WOT.

Figure 6 – Comparison between NAS without Training [9] and TE-NAS [28] search methods, both evaluated using the NAS-Bench-201 benchmark [24].



Architecture	**CIFAR-10**	**CIFAR-100**	**ImageNet-16-120**	**Search Cost (GPU sec.)**	**Search Method**
ResNet (He et al., 2016)	93.97	70.86	43.63	-	-
NAS w.o. Training (Mellor et al., 2020)	91.78(1.45)	67.05(2.89)	37.07(6.39)	4.8	training-free
TE-NAS (ours)	**93.9(0.47)**	**71.24(0.56)**	**42.38(0.46)**	1558	training-free
Optimal	94.37	73.51	47.31	-	-

Source: [28].

3 CONTRIBUTIONS

In this chapter, a novel trainingless NAS strategy is introduced, which is based on a new metric that aims at reducing the search time for effective network architectures without training. The proposed metric is based on the aforementioned NAS-WOT metric [9], due to its simplicity. Thus, in this chapter, the NAS-WOT metric is first analyzed in detail and, afterwards, the proposed metric is introduced.

3.1 EXAMINING THE NAS-WOT METRIC

As described in Section 2.5.1, the NAS-WOT metric [9] is based on ReLU activation patterns observed at the initialization of a candidate network. More specifically, the candidate network is exposed to a mini-batch of input data $X = \{\mathbf{x}_i\}_{i=1}^{K}$ and then, for each input sample $\mathbf{x}_i$, a binary activation vector $\mathbf{c}_i$ is obtained. From the Hamming distances between the obtained activation vectors, the $\mathbf{K}_\mathsf{H}$ matrix described in (2) is built. Finally, the NAS-WOT metric is obtained by evaluating the logarithm of the determinant of such a matrix (see (3)).

Aiming to obtain more insights regarding the NAS-WOT metric, we start by noticing that $\mathbf{K}_\mathsf{H}$ is a $K \times K$ matrix with the element at the i-th row and j-th column given by

$$k_{\mathsf{H}(i,j)} = N_\mathsf{A} - d_\mathsf{H}(\mathbf{c}_i,\mathbf{c}_j) \tag{4}$$

with K representing the size (number of samples) of the mini-batch, N_A the number of ReLU activations of the candidate network, and $d_\mathsf{H}(\mathbf{c}_i,\mathbf{c}_j)$ the Hamming distance between the i-th and the j-th binary activation vectors. Since the Hamming distance corresponds to the number of places where two binary arrays differ, one can easily conclude that (4) results in the number of places that have equal values when $\mathbf{c}_i$ and $\mathbf{c}_j$ are compared in a element-wise fashion. Part of this information can be obtained from the inner product $\mathbf{c}_i^\mathsf{T}\mathbf{c}_j$, which results in the number of places that have ones and are equal in $\mathbf{c}_i$ and $\mathbf{c}_j$. Moreover, by defining the complementary vector

$$\bar{\mathbf{c}}_i = \mathbf{1} - \mathbf{c}_i, \tag{5}$$

one can obtain, from the inner product $\bar{\mathbf{c}}_i{}^\mathsf{T}\bar{\mathbf{c}}_j$, the number of places that have zeros and are equal in $\mathbf{c}_i$ and $\mathbf{c}_j$. Thus, (4) can be rewritten as

$$k_{\mathsf{H}(i,j)} = \mathbf{c}_i^\mathsf{T}\mathbf{c}_j + \bar{\mathbf{c}}_i^\mathsf{T}\bar{\mathbf{c}}_j. \tag{6}$$

Now, by arranging the binary activation vectors $\mathbf{c}_1$ to $\mathbf{c}_K$ as columns of a $N_\mathsf{A} \times K$ matrix, one obtains

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_K \end{bmatrix}. \tag{7}$$

By arranging the complementary binary activation vectors $\bar{\mathbf{c}}_1$ to $\bar{\mathbf{c}}_K$ in a similar way, one has

$$\bar{\mathbf{C}} = \begin{bmatrix} \bar{\mathbf{c}}_1 & \bar{\mathbf{c}}_2 & \cdots & \bar{\mathbf{c}}_K \end{bmatrix}. \tag{8}$$

Finally, considering (4) to (8), matrix $\mathbf{K}_\mathsf{H}$ can be rewritten as

$$\mathbf{K}_\mathsf{H} = \mathbf{C}^\mathsf{T}\mathbf{C} + \bar{\mathbf{C}}^\mathsf{T}\bar{\mathbf{C}}. \tag{9}$$

3.1.1 Positive Semidefiniteness of $\mathbf{K}_\mathsf{H}$

According to [29] (Fact 3.7.25), the matrix product $\mathbf{A}^\mathsf{T}\mathbf{A}$ results in a $m \times m$ positive semidefinite matrix for any real matrix $\mathbf{A}$ with dimensions $n \times m$. From this fact, one can conclude that both $\mathbf{C}^\mathsf{T}\mathbf{C}$ and $\bar{\mathbf{C}}^\mathsf{T}\bar{\mathbf{C}}$ result in positive semidefinite matrices with dimensions $K \times K$. Moreover, from the definition of a positive semidefinite matrix [29], we have $\mathbf{v}^\mathsf{T}\mathbf{C}^\mathsf{T}\mathbf{C}\mathbf{v} \geq 0$ and also $\mathbf{v}^\mathsf{T}\bar{\mathbf{C}}^\mathsf{T}\bar{\mathbf{C}}\mathbf{v} \geq 0$ for any nonzero vector $\mathbf{v}$ with dimensions $K \times 1$. Thus, considering (9), we can write

$$\mathbf{v}^\mathsf{T}\mathbf{C}^\mathsf{T}\mathbf{C}\mathbf{v} + \mathbf{v}^\mathsf{T}\bar{\mathbf{C}}^\mathsf{T}\bar{\mathbf{C}}\mathbf{v} = \mathbf{v}^\mathsf{T}(\mathbf{C}^\mathsf{T}\mathbf{C} + \bar{\mathbf{C}}^\mathsf{T}\bar{\mathbf{C}})\mathbf{v} = \mathbf{v}^\mathsf{T}\mathbf{K}_\mathsf{H}\mathbf{v} \geq 0, \tag{10}$$

which proves that $\mathbf{K}_\mathsf{H}$ is also a positive semidefinite matrix.

3.1.2 Role of the Eigenvalues of $\mathbf{K}_\mathsf{H}$

Considering that $\mathbf{K}_\mathsf{H}$ is symmetric and positive semidefinite, a straightforward conclusion is that its eigenvalues are all real and greater than or equal to zero. This conclusion is important for understanding the NAS-WOT metric, since this metric depends on the determinant of $\mathbf{K}_\mathsf{H}$, which corresponds to the product of its eigenvalues, i.e.,

$$|\mathbf{K}_\mathsf{H}| = \det(\mathbf{K}_\mathsf{H}) = \prod_{k=i}^{K} \lambda_k \tag{11}$$

with λ_k representing the k-th eigenvalue of $\mathbf{K}_\mathsf{H}$. If all eigenvalues are positive or zero, we know that $|\mathbf{K}_\mathsf{H}|$ will be always greater than or equal to zero. Moreover, if only one eigenvalue is equal to zero (i.e., $\mathbf{K}_\mathsf{H}$ is rank-deficient), (11) will result in zero and the NAS-WOT metric from (3) will tend to $-\infty$. In fact, this situation can happen if the ReLU activation patterns are the same for two different samples of the mini-batch. As a result, both $\mathbf{C}$ and $\bar{\mathbf{C}}$ will have two columns that are equal, ultimately resulting in two similar columns/rows at the symmetric matrix $\mathbf{K}_\mathsf{H}$. This could be a problem for the NAS-WOT metric, since a promising architecture may end up being discarded due to a poor choice of two of the samples of the mini-batch. However, practical characteristics of the NAS-WOT process tend to eliminate such a problem, as discussed in the next section.

3.1.3 Practical Aspects of the NAS-WOT Metric

An important characteristic of practical NAS-WOT applications is that $N_\text{A} \gg K$, i.e., the number of ReLU activations in the candidate networks is much larger than the number of elements at the mini-batch. Fig. 7 presents histograms of number of ReLU activations for the architectures present in the four benchmarks considered for studying trainingless NAS (NATS-Bench TSS, NATS-Bench SSS, NASBench101, and NDS Facebook). From this figure, one can notice that the architectures under evaluation are generally very complex deep neural networks. As a consequence, $\mathbf{K}_\text{H}$ will normally be a relatively small matrix (since it is a $K \times K$ matrix, and $K \ll N_\text{A}$) with very large values at its main diagonal (since the main-diagonal elements of $\mathbf{K}_\text{H}$ are all equal to N_A).

The first important conclusion we can draw out of the aforementioned characteristics of $\mathbf{K}_\text{H}$ is that it will generally have relatively large eigenvalues. This conclusion is confirmed by the fact that the trace of $\mathbf{K}_\text{H}$ (sum of its eigenvalues) is equal to the product KN_A, with K in the range of hundreds and N_A in the range of tens of thousands to millions. This results in an average eigenvalue value equal to N_A (i.e., tens of thousands to millions). In fact, typical values for the NAS-WOT metric (which correspond to the natural logarithm of the product of the eigenvalues of $\mathbf{K}_\text{H}$) are larger than 1000, meaning that the actual product of the eigenvalues is typically larger than 1.97×10^{434}.

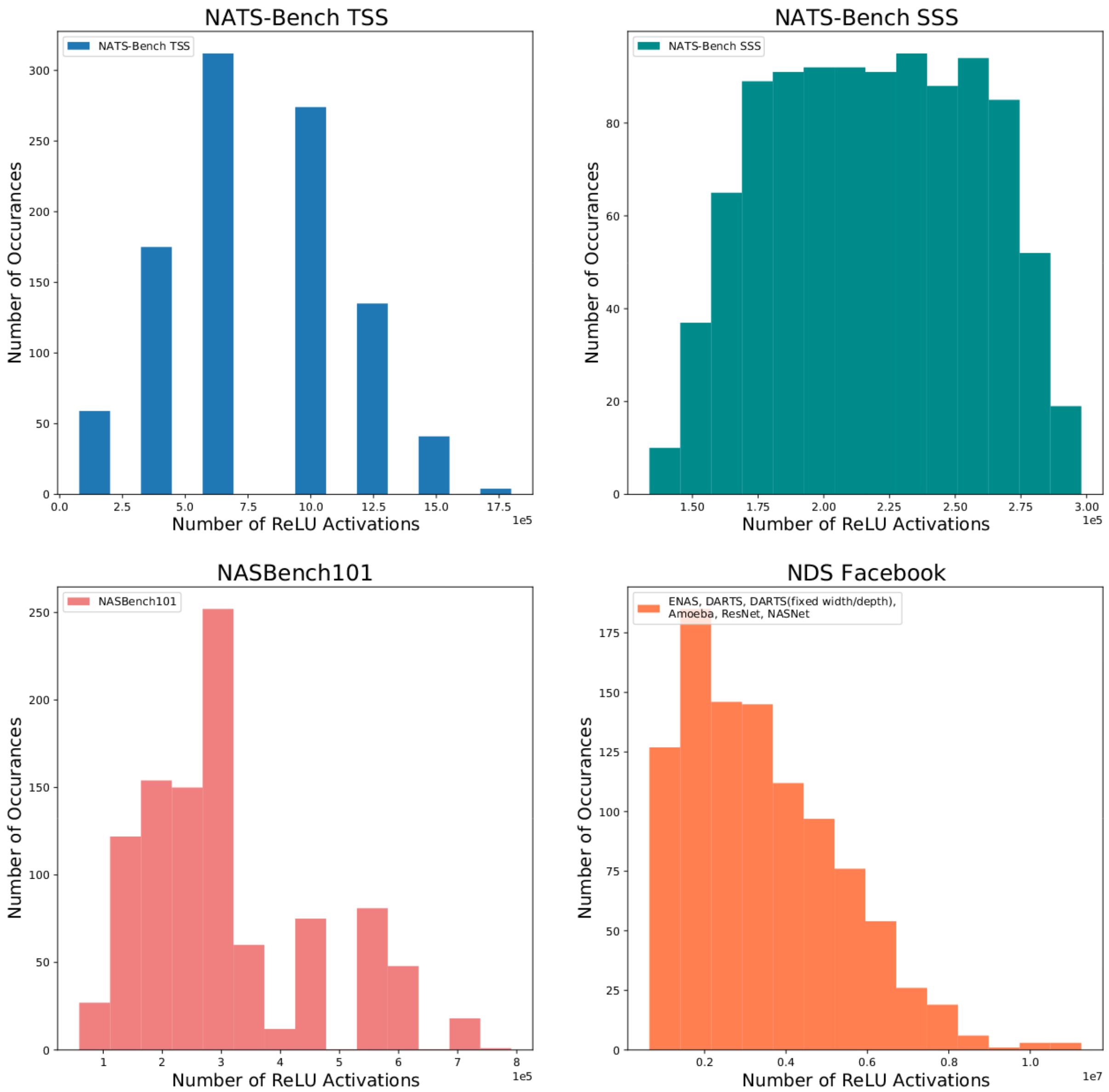
Another important practical conclusion regarding $\mathbf{K}_\text{H}$ is that it will normally be a positive definite matrix. This is due to the fact that, for $\mathbf{K}_\text{H}$ to be rank-deficient, two samples at the mini-batch must result in exactly the same ReLU activation patterns. However, the number of mini-batch samples is in the hundreds (typically equal to 128 in the papers from the literature), whereas each mini-batch sample will result in a ReLU activation pattern composed of tens of thousands or even millions of elements. In this context, the chance of two of these patterns to be equal tends to zero. In fact, it was also observed in practice that the eigenvalues are normally larger than 1, which is indeed expected considering all characteristics of $\mathbf{K}_\text{H}$ discussed here.

3.2 PROPOSED NAS METRIC

By using the properties of the logarithm operator, the NAS-WOT metric given in (3) can be rewritten as

$$\begin{aligned} s = \log(|\mathbf{K}_\text{H}|) &= \log(\det(\mathbf{K}_\text{H})) \\ &= \log(\prod_{k=i}^{K} \lambda_k) \\ &= \sum_{k=i}^{K} \log(\lambda_k). \end{aligned} \tag{12}$$

Figure 7 – Histograms of number of ReLU activations for the benchmarks considered in this work, obtained by randomly sampling 1000 network architectures.



Source: The Author.

From (12) and considering the characteristics of the eigenvalues of $\mathrm{K_H}$ discussed in the previous section, we can conclude that the NAS-WOT metric will be given by the sum of the logarithms of the (usually) very large values of the eigenvalues of $\mathrm{K_H}$. In this context, we assume here that a fair estimation for such a metric can be obtained by removing the logarithm operator from the elements in the summation in (12). In spite of being a rough estimation, our hypothesis is that it will be reasonable for trainingless NAS where the main purpose is to eliminate bad candidates. Thus, considering such an estimation, (12) becomes

$$\begin{aligned} s_{\mathrm{p}} &= \sum_{k=i}^{K} \lambda_k \\ &= \mathrm{trace}(\mathbf{K}_{\mathrm{H}}) \\ &= N_{\mathrm{A}} \times K. \end{aligned} \tag{13}$$

Note from (13) that, by adopting the aforementioned estimation, the proposed metric becomes the trace of $\mathbf{K}_{\mathrm{H}}$, which in turn is known to be equal to $N_{\mathrm{A}} \times K$. This makes the proposed metric mathematically simple to be evaluated, since it basically corresponds to the number of ReLU activations in the candidate network (N_{A}) multiplied by a constant. It is worth mentioning, however, that obtaining N_{A} is not as simple as it seems, due to the complexity of the candidate networks that are usually present in NAS search spaces. Despite this, after obtaining N_{A}, a lot of calculations are avoided, saving important search time.

In [9], the authors of the NAS-WOT metric show some concerns regarding the robustness of their metric regarding practical aspects as the samples at the mini-batch, the mini-bath size, the network initializations, and the evolution as the network is trained. In that work, an ablation study is conducted to test the NAS-WOT metric regarding these aspects. The metric proposed here, in contrast, is naturally robust to all this practical questions, since it depends on the structure of the network (the mini-batch size is merely a constant that affects all candidate architectures equally).

Another important aspect that of the proposed metric is that, by dropping the logarithm operator in the summation of eigenvalues, it ends up giving more weight to larger eigenvalues as compared to the NAS-WOT metric. In other words, larger eigenvalues tend to dominate the proposed metric. This, however, does not seem to be an important issue, as demonstrated in the practical results presented in the next chapter of this work.

3.3 PROPOSED METRIC AS A COMPUTATIONAL COMPLEXITY PREDICTOR

One important aspect of neural network architectures is the computational complexity required for their implementation for inference (after training). This is particularly important for embedded implementations of neural networks, as illustrated by the μNAS case (see section 2.4). The proposed metric is directly related with the total number of ReLU activations in the network and, thus, it in general tends to be a good indicator of computational complexity for inference. Considering this fact and also that a trade-off between complexity and accuracy is often pursued in practice, another proposal in this dissertation is to use penalized versions of the proposed metric aiming at accounting for both accuracy and inference complexity together. The idea is to obtain a metric that implicitly takes into account complexity restrictions when ranking the candidate architectures. Some of the experimental results presented in the next chapter address this approach.

4 RESULTS AND DISCUSSIONS

In this chapter, experimental results are presented aiming to assess the effectiveness of the proposed metric in the context of NAS. Such results were obtained considering publicly-available benchmarks that were developed aiming to help standardizing NAS research works. These benchmarks contain tractable search spaces and metadata regarding the training of the involved neural network architectures as, for instance, the accuracy obtained after training each candidate architecture. Thus, the benchmarks considered in this work are the NATS-Bench TSS [24], NATS-Bench SSS [25], NASBench101 [26], and NDS Facebook [27]. Considering such benchmarks, the experiments conducted for this dissertation are based in three classic datasets: CIFAR-10 [30], CIFAR-100 [31], and ImageNet16-120 [32].

4.1 EVALUATING THE PROPOSED METRIC

4.1.1 Experiment 1

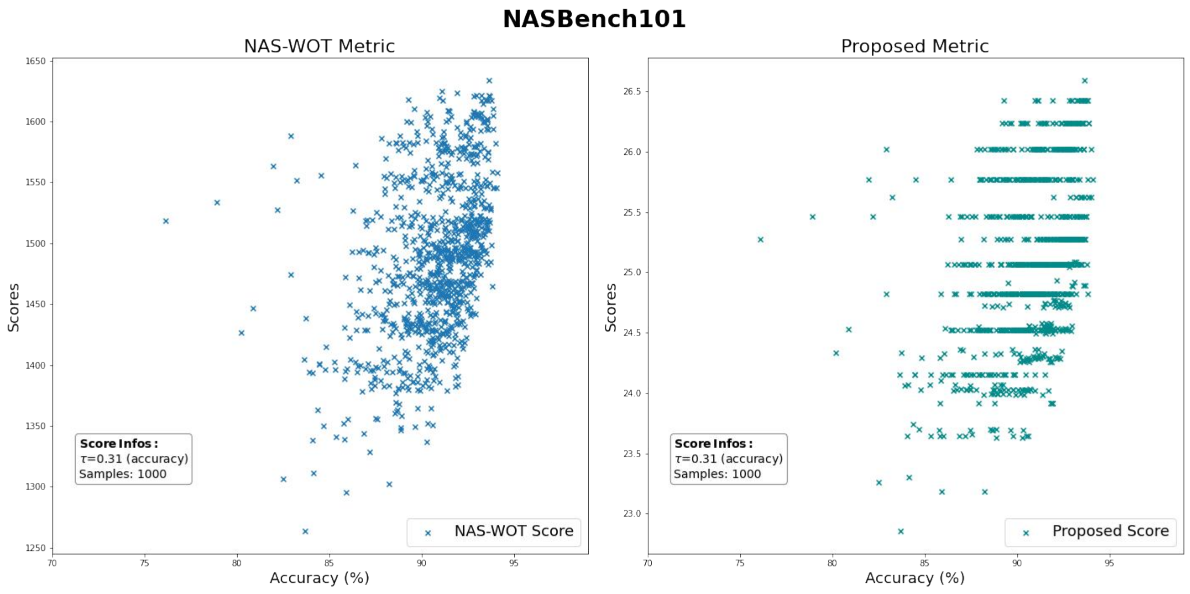
Following the methodology adopted in [9], the first experiment for evaluating the effectiveness of the proposed metric is carried out by randomly sampling a thousand neural architectures for each considered dataset in each considered benchmark. Both the NAS-WOT metric and the proposed metric are evaluated for each architecture and, then, the ontained values plotted against the corresponding accuracies available in the benchmark. The obtained plots provide an important idea if whether or not a metric is a good predictor for the accuracy of the network. Such aspect is also evaluated using the Kendall's τ correlation coefficient, which is defined as

$$\tau = \frac{\text{(number of concordant pairs)} - \text{(number of discordant pairs)}}{\frac{n(n-1)}{2}}, \tag{14}$$

with n denoting the total number of evaluated points.

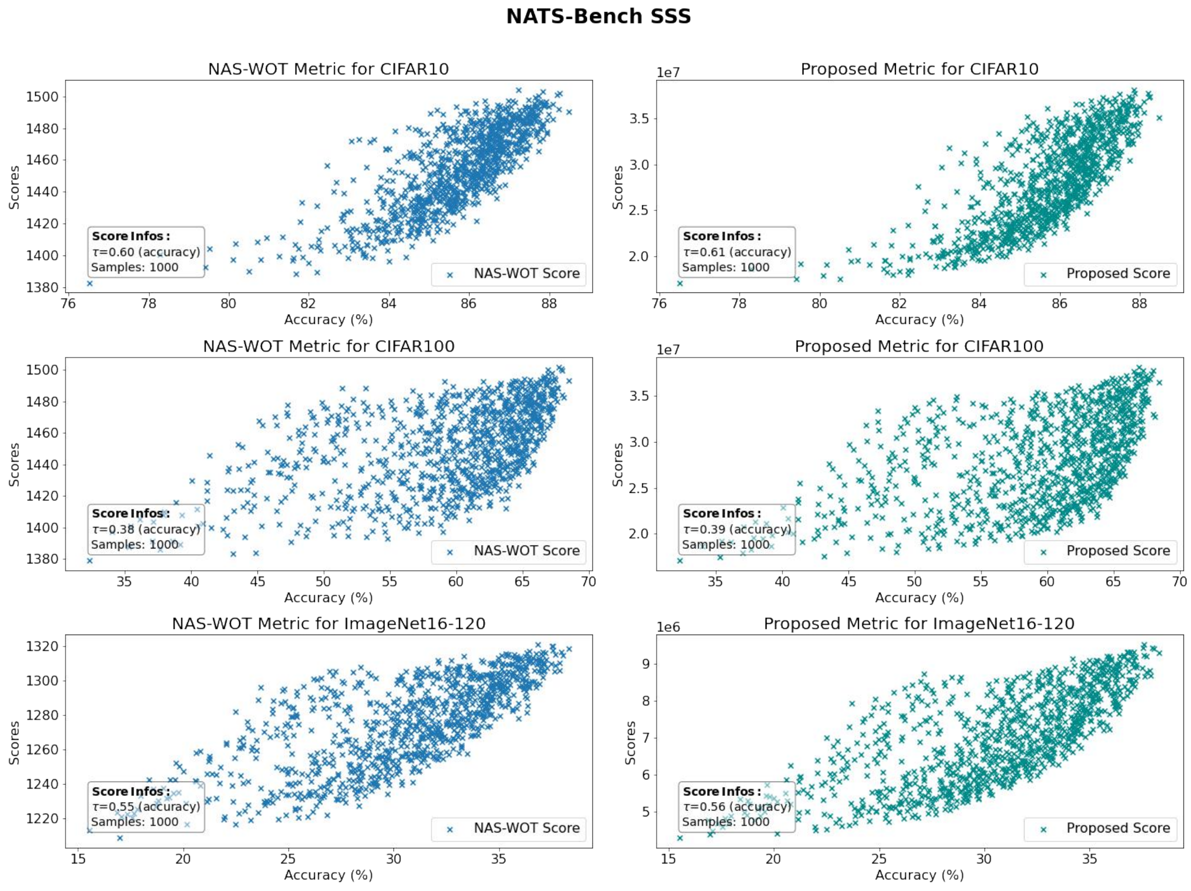
The obtained results are presented in Fig. 8 for the NAS-Bench101 benchmark, Fig. 9 for the NATS-Bench SSS, Fig. 10 for the NATS-Bench TSS, and Fig. 11 for the NDS Facebook. In all cases, one can notice a great similarity between the NAS-WOT and the proposed metric, with some difference observed only for the NATS-Bench TSS benchmark. This is due to the particular characteristics of such a benchmark and the related search space, which involves large groups of networks having similar numbers of ReLU activations. This characteristic can also be observed from Fig. 7. The Kendall's τ results for all cases are also summarized in Fig. 13. In general, these results show that the proposed metric, in spite of being very simple, is a very good proxy for the NAS-WOT metric, with the potential for producing similar results in a much smaller time span.

Figure 8 – Plots of the proposed metric and the NAS-WOT metric for 1000 randomly sampled untrained neural networks architecture in NAS-Bench-101 against the validation accuracy. The scores were calculated for the CIFAR-10 dataset.



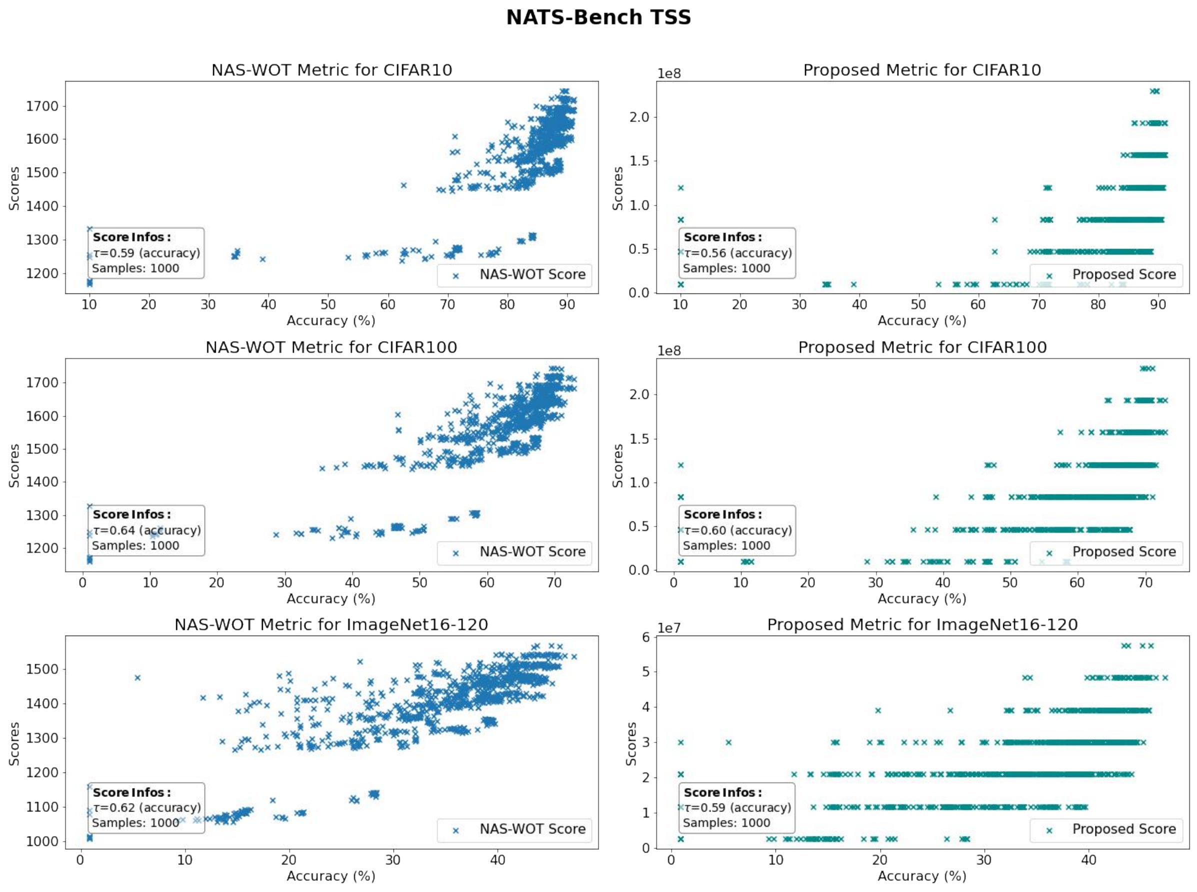
Source: The Author.

Figure 9 – Plots of the proposed metric and the NAS-WOT metric for 1000 randomly sampled untrained neural networks architecture in NATS-Bench SSS against the validation accuracy. The scores were calculated for the CIFAR-10, CIFAR-100 and ImageNet16-120 datasets.



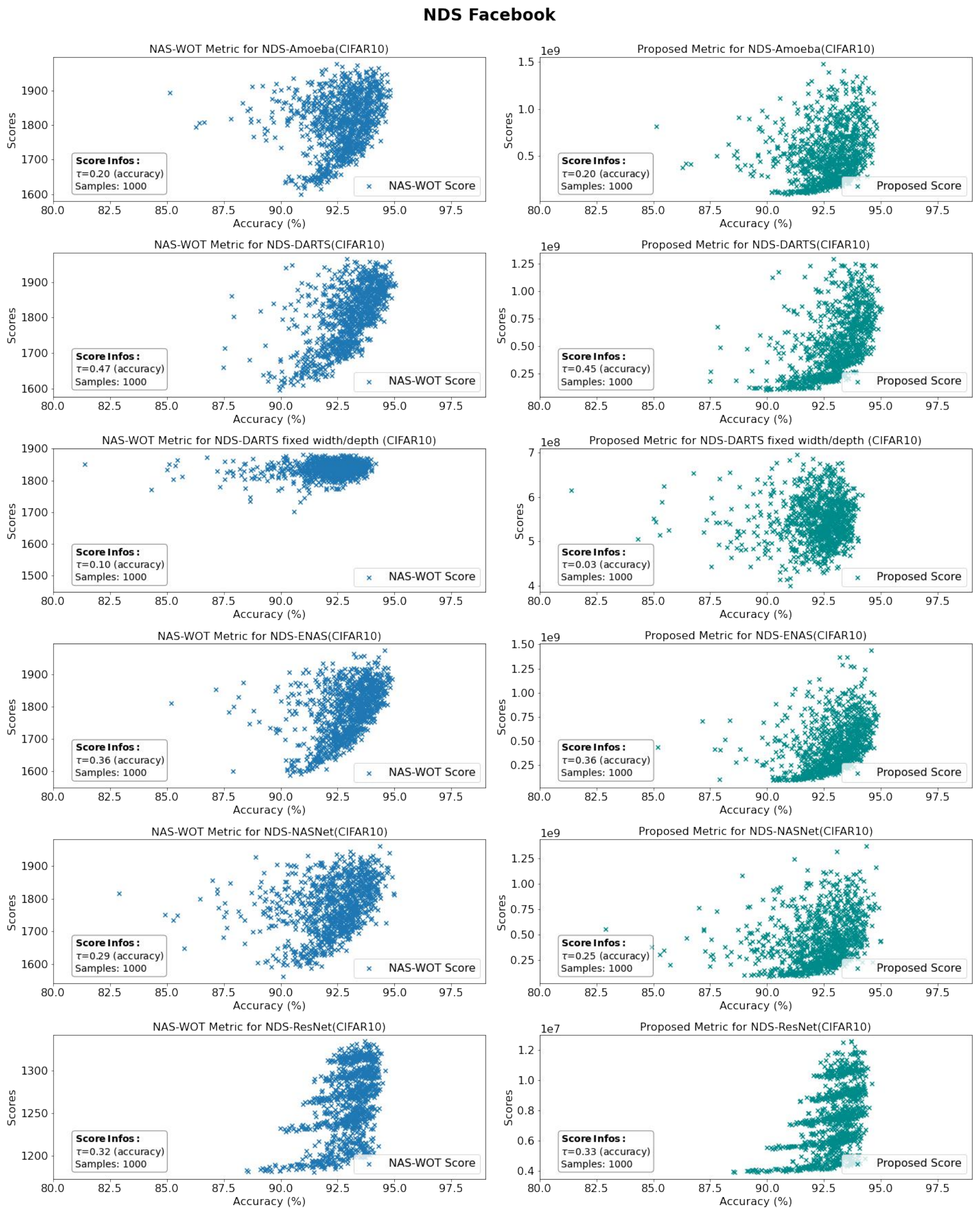
Source: The Author.

Figure 10 – Plots of the proposed metric and the NAS-WOT metric for 1000 randomly sampled untrained neural networks architecture in NATS-Bench TSS against the validation accuracy. The scores were calculated for the CIFAR-10, CIFAR-100 and ImageNet16-120 datasets.



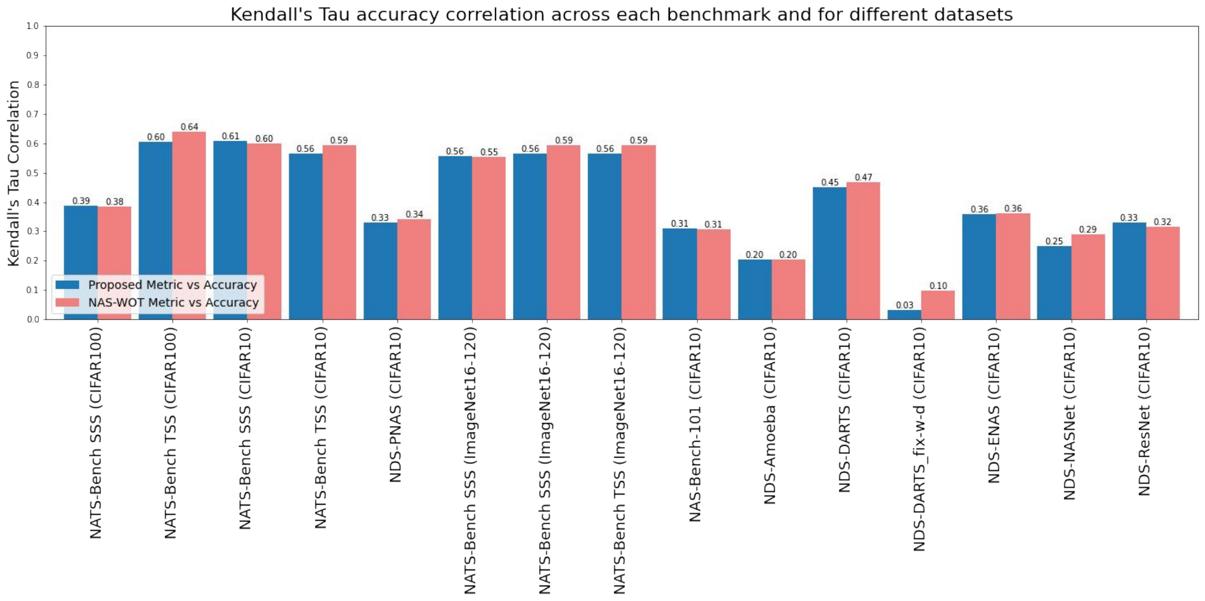
Source: The Author.

Figure 11 – Plots of the proposed metric and the NAS-WOT metric for 1000 randomly sampled untrained neural networks architecture in NDS-Amoeba, NDS-DARTS, NDS-DARTS fixed width/depth, NDS-ENAS, NDS-NASNet and NDS-ResNet against the validation accuracy. The scores were calculated for the CIFAR-10 dataset.



Source: The Author.

Figure 12 – Kendall's τ correlation for different search spaces and datasets for both proposed and NAS-WOT metrics.



Source: The Author.

Algorithm 1 Search Algorithm used to evaluate both Proposed and NAS-WOT score metric.

```
generator = RandomGenerator()
best_net, best_score = None, 0
for $i = 1 : N$ do
    net = generator.generate()
    score = net.score()
    if score > best_score then
        best_net, best_score = net, score
    end if
end for
chosen_net = best_network
```

Source: [9]

Table 4 – Google Colab Pro [34] hardware used to evaluate all experiments, containing information about the CPU and GPU.

Computer Specs		GPU Spec	
Processor	Intel(R) Xeon(R)	**GPU**	Tesla P100
CPU Freq.	2.20GHz	**GPU Memory**	16GB
No. CPU Cores	2	**Single-Precision Performance**	10.6 TFLOPS
CPU Family	Haswell	**Double-Precision Performance**	5.3 TFLOPS
Available RAM	16GB		

Source: The Author.

4.1.2 Experiment 2

In the second experiment, the idea is to build a complete NAS system. To this end, random search [33] is used for exploring the search spaces defined in the considered benchmarks. The pseudo algorithm described as Algorithm 1 is then adopted, which corresponds basically the algorithm used in [9]. In all experiments, such an algorithm is run for 10 times and the average and standard deviance of the results are obtained. The specifications of the computer used to run the experiments are described in Table 4. The computer is from the Google Colab Pro services [34]. To ensure the veracity and to obtain a fair comparison of the results, both the proposed score metric and NAS-WOT score metric were evaluated using the considered experimental setup.

The obtained results are presented in Table 5 together with results from the literature regarding other trainingless and non trainingless methods. For the non trainingless methods (non weight sharing and weight sharing), the reported results are those reported at were extracted from [9]. In contrast, for the trainingless methods, we have: i) results obtained using the NATS-Bench TSS and NATS-Bench SSS benchmarks; ii) search time and accuracy results for the NAS-WOT and proposed metrics obtained in our experiments; iii) the time results for TE-NAS and EPE-NAS estimated considering the relative percentage times with respect to the NAS-WOT from the respective works from the literature together with the search time obtained for the NAS-WOT in our experiments; and iv) accuracy results for TE-NAS and EPE-NAS obtained from the values presented in the respective works.

From the results presented in Table 5, one observes that the proposed method is the fastest one, while yielding results comparable to those obtained by other methods in terms of accuracy. Moreover, in the training-free block in the Table 5, a comparison between the proposed metric and the NAS-WOT is carried out by highlighting in boldface the best result between the two for each case considered. This comparison shows that in some cases the NAS-WOT finds better architectures in terms of accuracy. However, considering that the difference between the accuracies is small and also that the time spent running the search method is much lower for the proposed metric (see Table 6), we can claim that the proposed metric is a better choice for implementing a trainingless NAS system.

Moreover, the training-free block in the Table 5 has the bests results, between the proposed score and NAS-WOT score, highlighted in bold for the same number of networks evaluated (*N*). This shows that the proposed score outperforms NAS-WOT in the majority of the cases, both in terms of time spent searching and the accuracy of the top neural network found using a random search. The Table 6 shows how much faster (in percentage) the proposed score is compared to NAS-WOT.

Furthermore, the Table 5 shows that in some cases the NAS-WOT finds better architectures in terms of accuracy. But considering that the difference between the accuracies found is lower, if not equal, to the variance, and the time spent running the search method is lower for the proposed metric (see Table 6, it is possible to consider the proposed metric as the best performance estimator, with a little trade-off in terms of the accuracy found having the difference in the range of the variance.

4.2 SCALING THE PROPOSED METRIC BY NETWORK SIZE

From the results presented in Table 5 and also in Figs. 8, 9, and 10, one can notice that the proposed metric presents a relatively high correlation with accuracy for the networks belonging to the NATS-Bench SSS benchmark. Such a benchmark is in fact the part of the NATS-Bench benchmark involving a size search space (SSS), involving a major focus on varying the size of the architecture candidates. This leads to the insight that the proposed metric tends to be also a good predictor for network size, as mentioned in Section 3.3. To test this idea, The Kendall's τ correlation coefficient was calculated, considering different benchmarks, for the following cases: i) proposed metric versus network size; ii) NAS-WOT metric versus network size; and iii) proposed metric versus NAS-WOT metric. The obtained results, presented in Fig. 13, show that the proposed metric is in fact a better size predictor than the NAS-WOT metric for all

Table 5 – Comparison of several search methods evaluated using the NATS-Bench, using different numbers (*N*) of random samples for the trainingless methods. The performance is visualized in two forms, the time spent searching and the mean±std. accuracies. The highlights in bold represent the best value in the comparison between the proposed and NAS-WOT metrics. All scenarios were run at least 10 times.

Method	**Search (s)**	**CIFAR-10**		**CIFAR-100**		**ImageNet16-120**	
		validation	**test**	**validation**	**test**	**validation**	**test**
				Non-weight sharing			
REA	12000	91.19 ± 0.31	93.92 ± 0.30	71.81 ± 1.12	71.84 ± 0.99	45.15 ± 0.89	45.54 ± 1.03
RS	12000	90.93 ± 0.36	93.70 ± 0.36	70.93 ± 1.09	71.04 ± 1.07	44.45 ± 1.10	44.57 ± 1.25
REINFORCE	12000	91.09 ± 0.37	93.85 ± 0.37	71.61 ± 1.12	71.71 ± 1.09	45.05 ± 1.02	45.24 ± 1.18
BOHB	12000	90.82 ± 0.53	93.61 ± 0.52	70.74 ± 1.29	70.85 ± 1.28	44.26 ± 1.36	44.42 ± 1.49
				Weight sharing			
RSPS	7587	84.16 ± 1.69	87.66 ± 1.69	59.00 ± 4.60	58.33 ± 4.34	31.56 ± 3.28	31.14 ± 3.88
DARTS-V1	10890	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
DARTS-V2	29902	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
GDAS	28926	90.00 ± 0.21	93.51 ± 0.13	71.14 ± 0.27	70.61 ± 0.26	41.70 ± 1.26	41.84 ± 0.90
SETN	31010	82.25 ± 5.17	86.19 ± 4.63	56.86 ± 7.59	56.87 ± 7.77	32.54 ± 3.63	31.90 ± 4.07
ENAS	13315	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
				Training-free for NATS-Bench TSS			
NAS-WOT (N=10)	3.68	89.62 ± 0.33	87.18 ± 1.35	6.94 ± 3.09	57.65 ± 3.61	35.24 ± 19.49	35.43 ± 19.51
EPE-NAS (N=10)	2.73	89.90 ± 0.21	92.63 ± 0.32	69.78 ±	70.10 ± 1.71	41.73 ± 3.60	41.92 ± 4.25
Proposed Metric (N=10)	**2.28**	90.62 ± 0.93	89.9 ± 1.01	69.69 ± 3.85	69.17 ± 1.16	35.03 ± 2.96	35.27 ± 2.49
NAS-WOT (N=100)	45.93	90.39 ± 0.92	87.48 ± 0.81	71.07 ± 8.57	70.67 ± 3.07	35.54 ± 18.22	35.78 ± 18.64
EPE-NAS (N=100)	35.91	88.74 ± 3.16	91.59 ± 0.87	67.28 ± 3.68	67.19 ± 3.82	38.66 ± 4.75	38.80 ± 5.41
TE-NAS (N=100)	6232	93.9 ± 0.47	-	71.24 ± 0.56	-	42.38 ± 0.46	-
Proposed Metric (N=100)	**29.66**	91.0 ± 0.82	90.44 ± 0.28	70.91 ± 1.16	70.72 ± 12.76	46.26 ± 5.51	46.03 ± 5.59
NAS-WOT (N=1000)	307.19	90.44 ± 0.73	89.42 ± 0.46	68.81 ± 2.95	61.87 ± 10.27	43.95 ± 2.05	44.44 ± 2.10
EPE-NAS (N=1000)	250.76	87.87 ± 0.85	91.31 ± 1.69	69.44 ± 0.83	69.58 ± 0.83	41.86 ± 2.83	41.84 ± 2.06
Proposed Metric (N=1000)	**146.79**	90.95 ± 1.01	90.44 ± 0.31	73.51 ± 2.03	71.12 ± 2.66	47.08 ± 9.56	46.33 ± 2.05
				Training-free for NATS-Bench SSS			
NAS-WOT (N=10)	2.92	89.68 ± 0.51	88.57 ± 0.33	66.50 ± 9.67	62.3 ± 2.98	39.06 ± 0.02	38.83 ± 0.21
Proposed Metric (N=10)	**1.80**	90.55 ± 1.60	89.62 ± 0.29	68.90 ± 5.25	61.12 ± 4.34	42.69 ± 6.53	42.88 ± 6.04
NAS-WOT (N=100)	30.04	90.14 ± 0.30	89.38 ± 0.54	68.80 ± 6.14	68.06 ± 2.54	40.22 ± 3.73	40.48 ± 3.70
Proposed Metric (N=100)	**21.86**	92.66 ± 0.23	90.14 ± 0.22	70.72 ± 12.76	68.42 ± 4.47	46.27 ± 5.51	46.03 ± 6.67
NAS-WOT (N=1000)	251.63	92.14 ± 0.79	90.8 ± 0.31	66.53 ± 11.36	67.64 ± 0.0	44.57 ± 1.48	45.08 ± 1.55
Proposed Metric (N=1000)	**144.54**	93.14 ± 0.09	90.44 ± 0.3	73.51 ± 2.03	71.04 ± 10.34	47.07 ± 5.59	46.27 ± 5.54

Source: The Author and [9].

Table 6 – Comparison of how much faster is the proposed metric against the NAS-WOT metric, in percentage. Values compared for the NATS-Bench SSS and NATS-Bench TSS with *N* = 10, *N* = 100 and *N* = 1000 neural networks sampled.

	NATS-Bench TSS	**NATS-Bench SSS**
N=10	38.04%	38.35%
N=100	38.42%	27.23%
N=1000	52.21%	42.55%

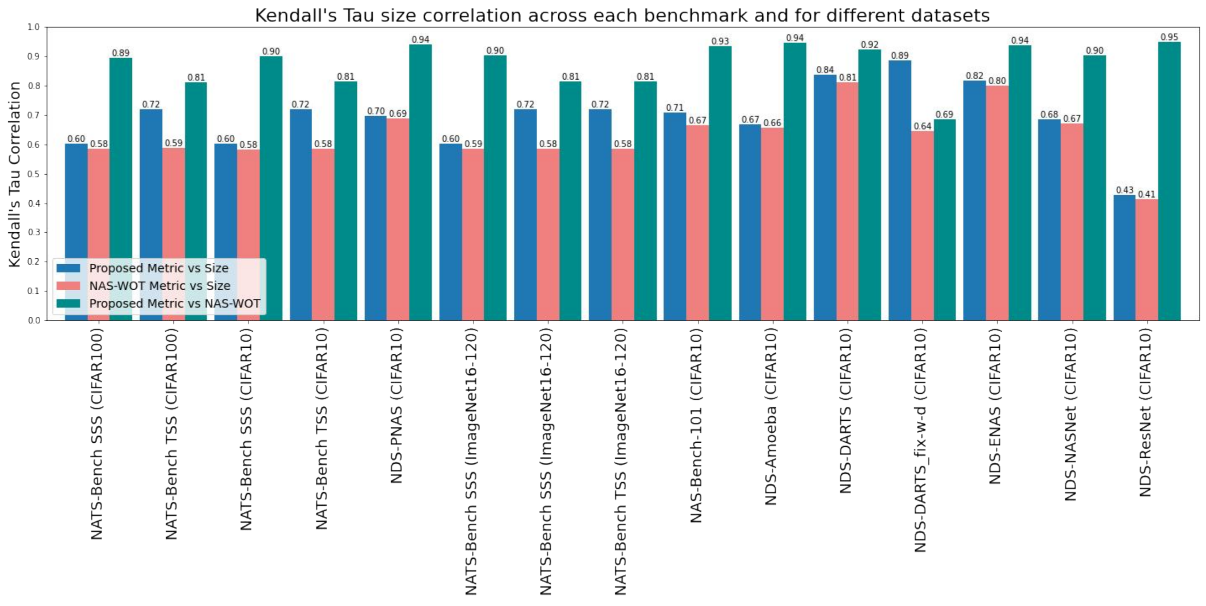
Source: The Author.

cases considered.

4.2.1 Experiment 3

Considering that the proposed metric tends to be a better predictor of network size, the focus in Experiment 3 is to test the idea, described in Section 3.3, of penalizing the proposed metric aiming to obtain a prediction of the trade-off between accuracy and network size. The functions considered for such a penalization are the Gaussian

Figure 13 – Kendall's τ correlation for i) proposed metric versus network size; ii) NAS-WOT metric versus network size; and iii) proposed metric versus NAS-WOT metric. Results obtained for different benchmarks/datasets.



Source: The Author.

Distribution Function (GDF), given by

$$f(x;\mu,\sigma) = \begin{cases} \dfrac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} & x \geq \mu \\ x & \text{otherwise,} \end{cases} \tag{15}$$

the Hyperbolic Secant Distribution Function (HSDF), given by

$$f(x;\mu) = \begin{cases} \dfrac{1}{2}\,\text{sech}\left(\dfrac{\pi}{2}(x-\mu)\right), & x \geq \mu \\ x, & \text{otherwise,} \end{cases} \tag{16}$$

and also the Rayleigh Distribution Function (RDF):

$$f(x;\mu,\sigma) = \begin{cases} \dfrac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}}, & x \geq \mu \\ x, & \text{otherwise.} \end{cases} \tag{17}$$

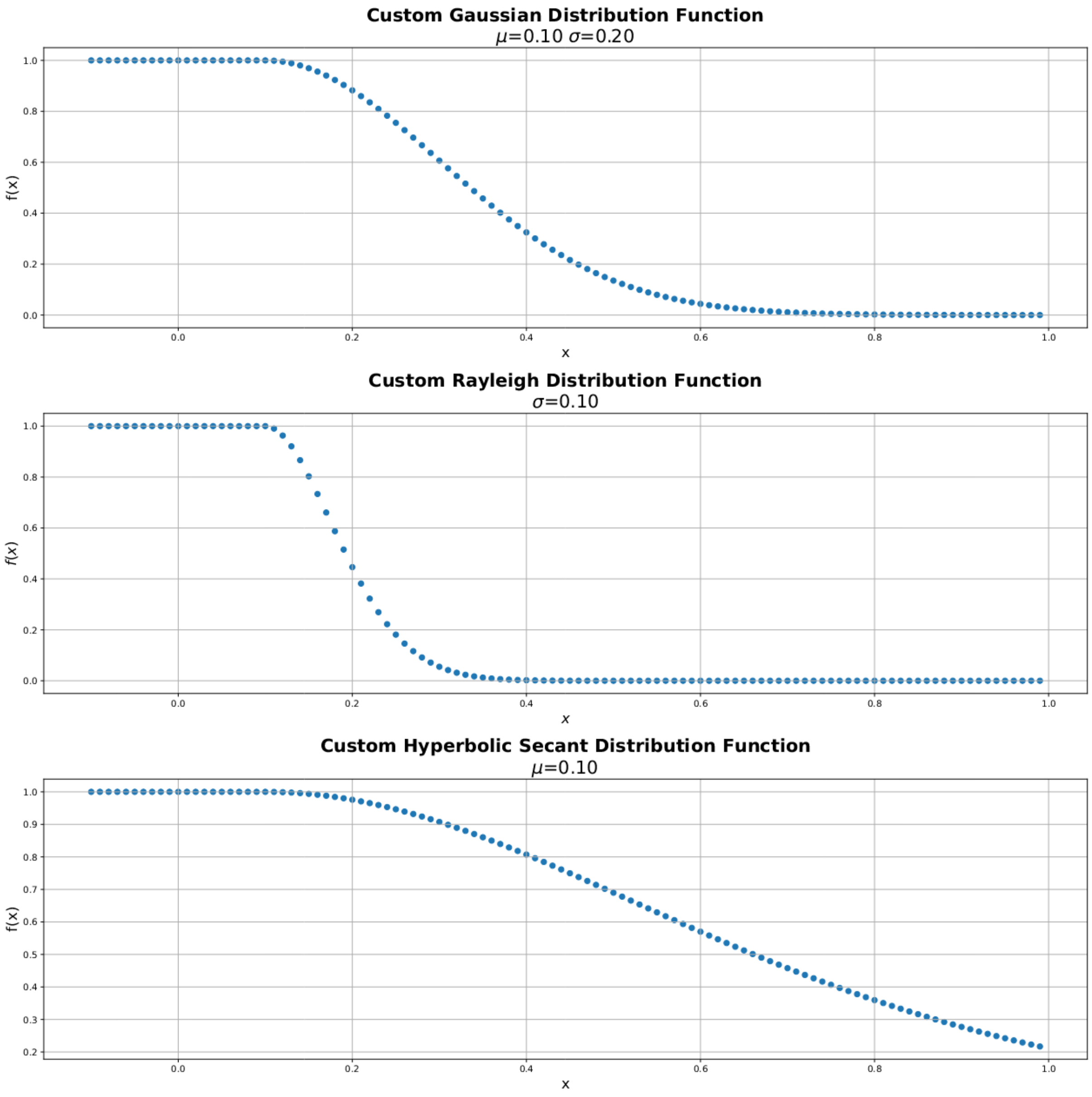
As illustrated in Fig. 14, all these functions are characterized for being equal to one at the origin ($x = 0$) and then vanishing (tending to zero) after a smooth transition at some point determined by their parameters. Thus, by tuning the transition position (related with the target network size), one can constrain the NAS process, avoiding very complex architectures.

Figs. 15 and 16 presents the results of the GDF-penalized metric for the NATS-Bench SSS and NATS-Bench TSS benchmarks, respectively. In light green, one has the original values for the proposed metric, whereas in orange one has the GDF-penalized metric. The mean value μ and the standard deviation σ^2 were chosen respectively as 0.2 and 0.4, targeting networks of up to 200 kB in size. From these results, we can clearly see the effect of the penalization, resulting in larger score around the 200 kB region. Figs. 17 and 18 show similar results considering the HSDF-penalized metric, whereas Figs. 19 and 20 also show similar results for the RDF-penalized metric.

4.2.2 Experiment 4

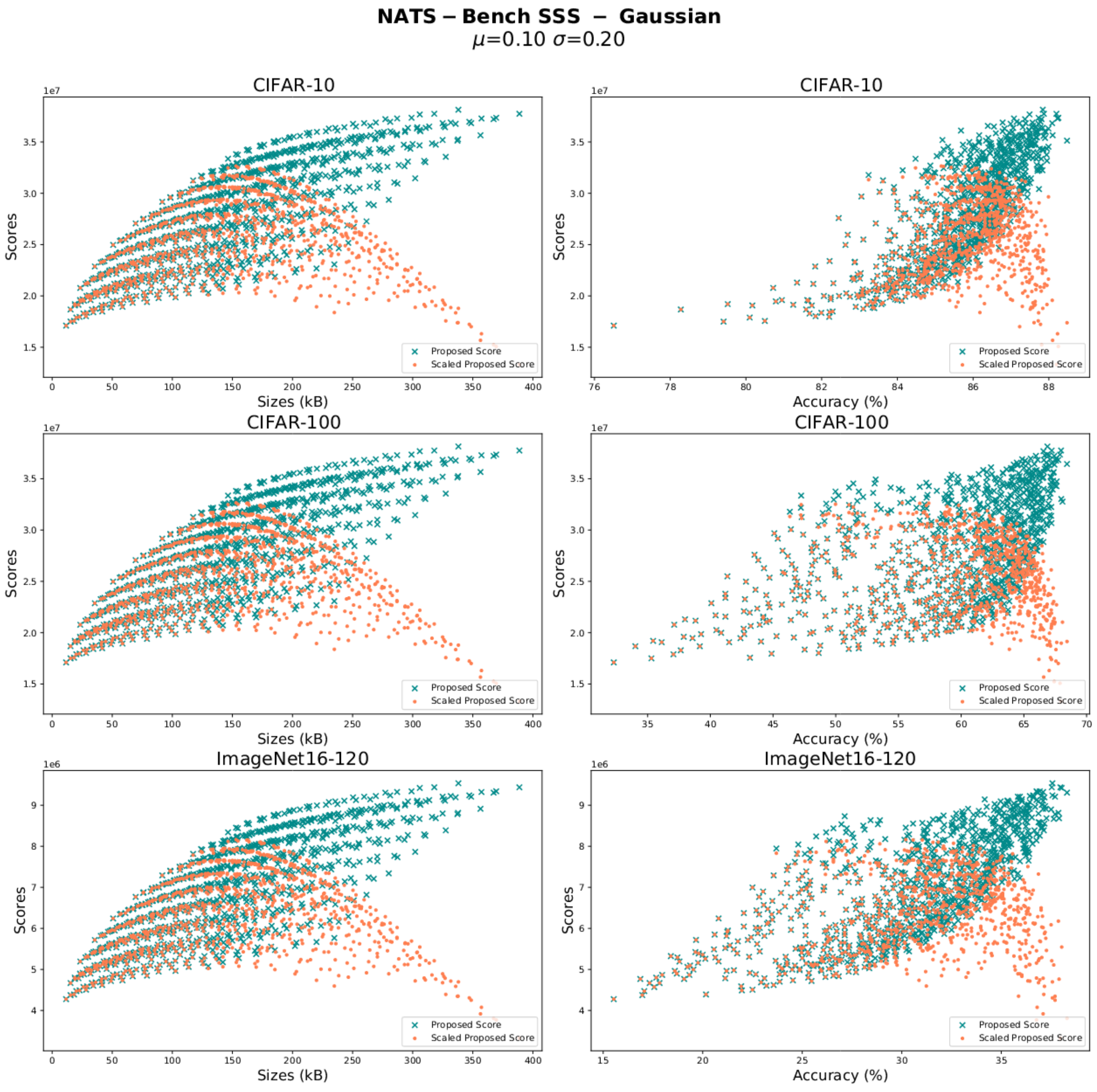
Now that the idea of penalizing the proposed metric has shown some promising results from a more theoretical standpoint, the purpose of Experiment 4 is to evaluate if it can be really effective in the context of practical NAS. To this end, we use the GDF-penalized version of the proposed metric together with the NATS-Bench TSS and NATS-Bench SSS benchmarks. The obtained results, shown in Table 7, demonstrate the effectiveness of the penalized version of the proposed metric for obtaining networks with a very good trade-off between accuracy and computational complexity.

Figure 14 – The different penalization functions considered: GDF on top, RDF in the middle and HSDF on the bottom.



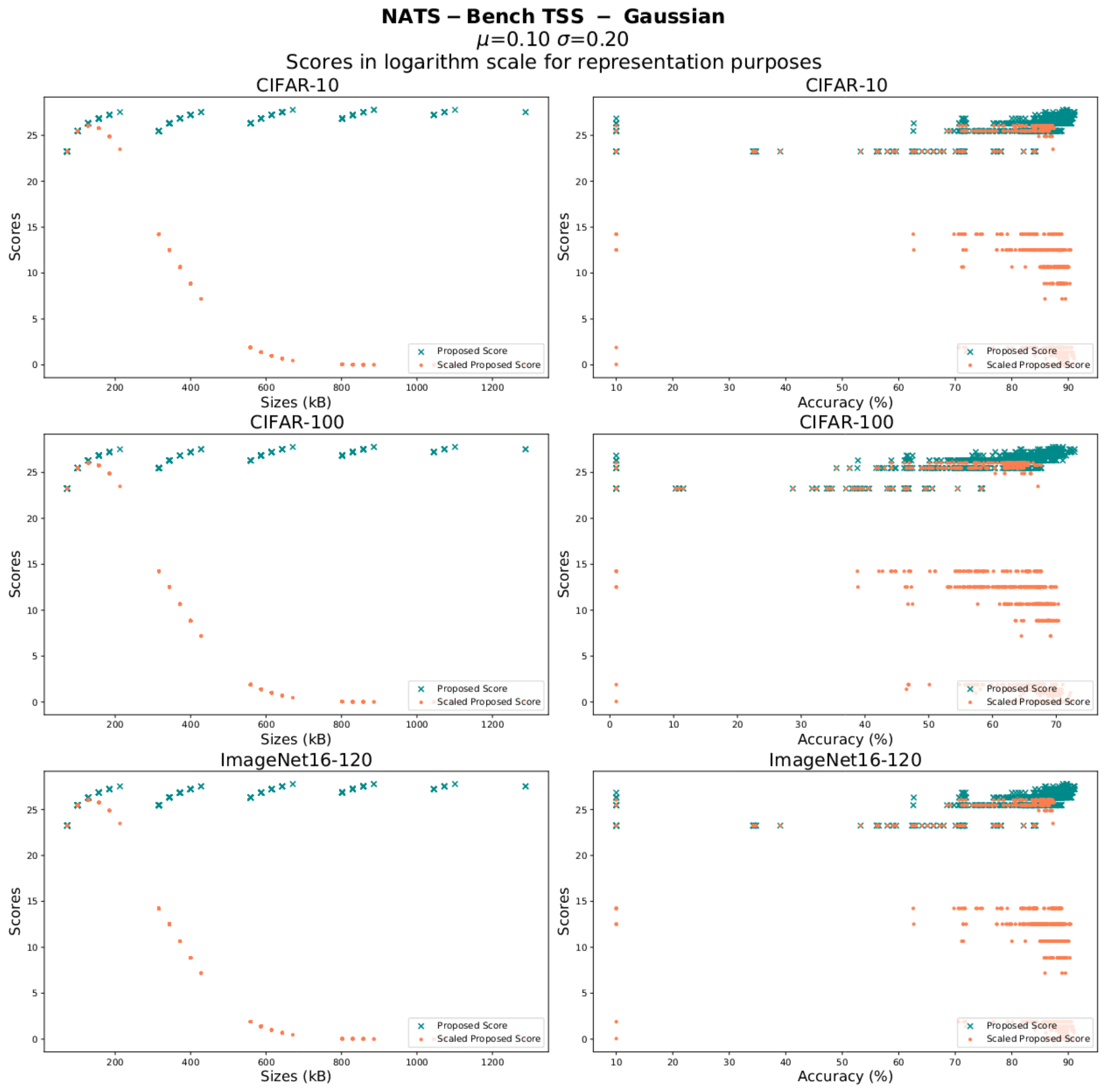
Source: The Author.

Figure 15 – Plots of the proposed metric in NATS-Bench TSS search space with and without scaling using Gaussian Distribution as penalty function for network size values superior to 200kB.



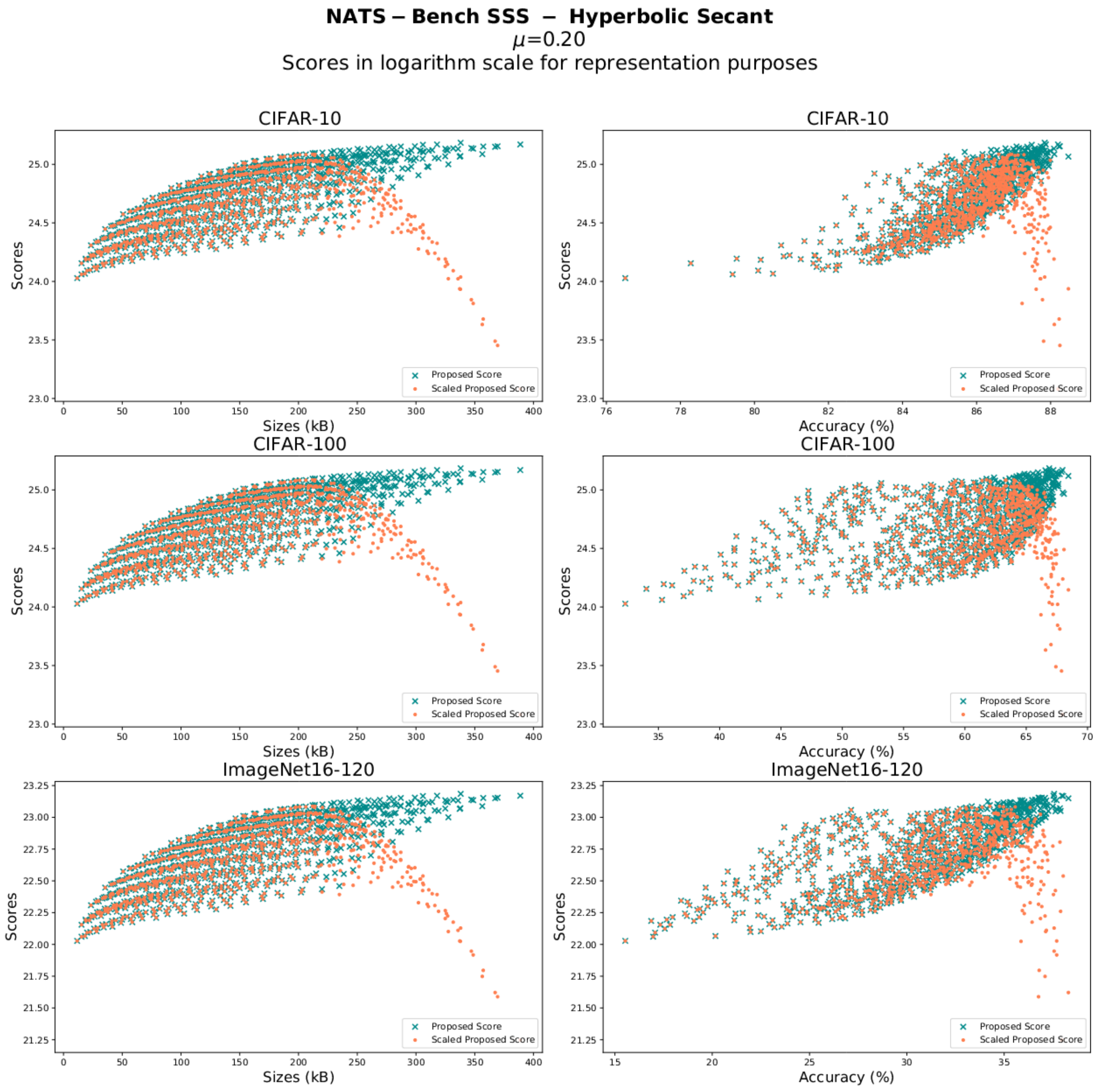
Source: The Author.

Figure 16 – Plots of the proposed metric in NATS-Bench TSS search space with and without scaling using Gaussian Distribution as penalty function for network size values superior to 200kB.



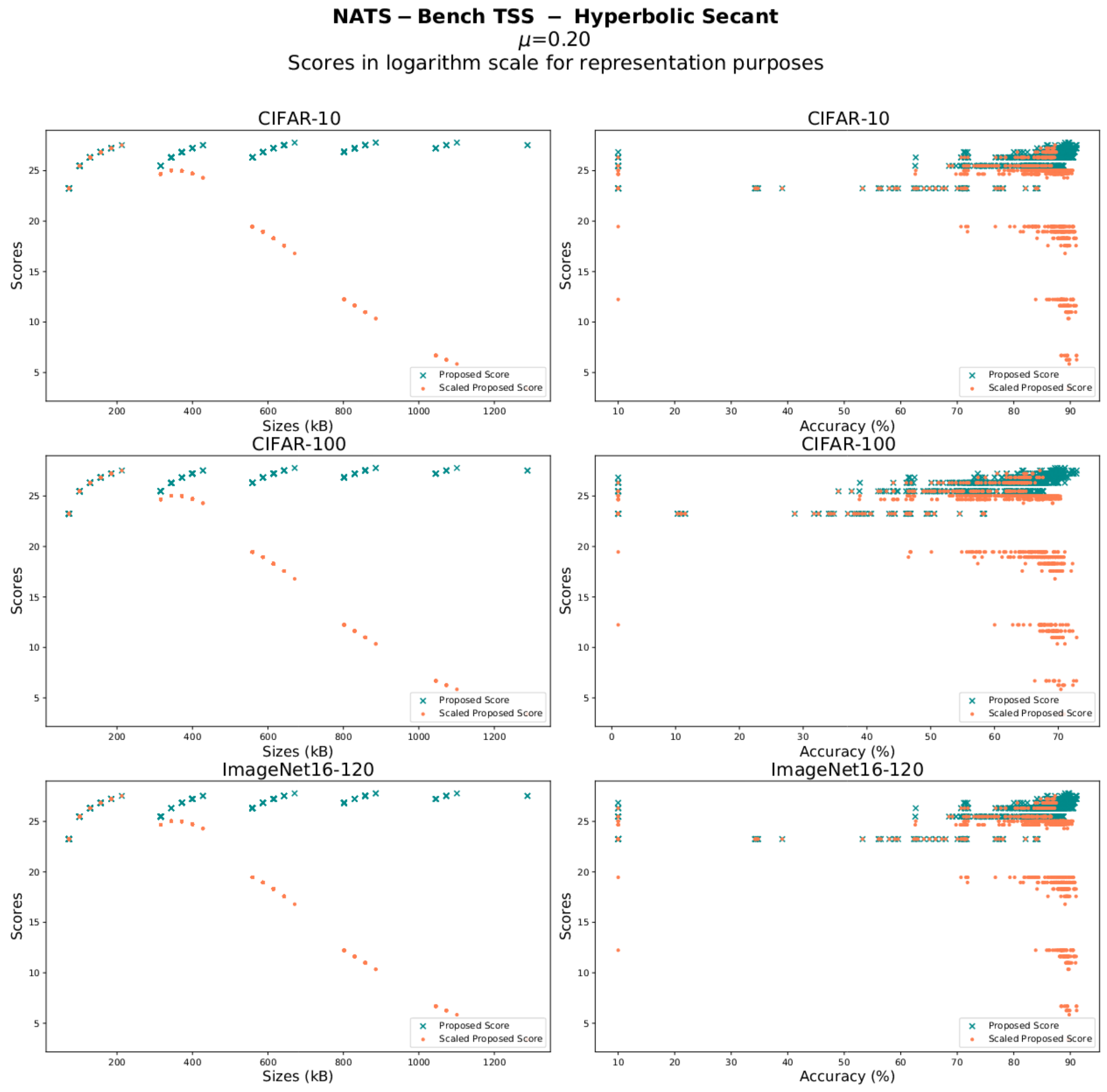
Source: The Author.

Figure 17 – Plots of the proposed metric in NATS-Bench SSS search space with and without scaling using Hyperbolic Secant Distribution as penalty function for network size values superior to 200kB.



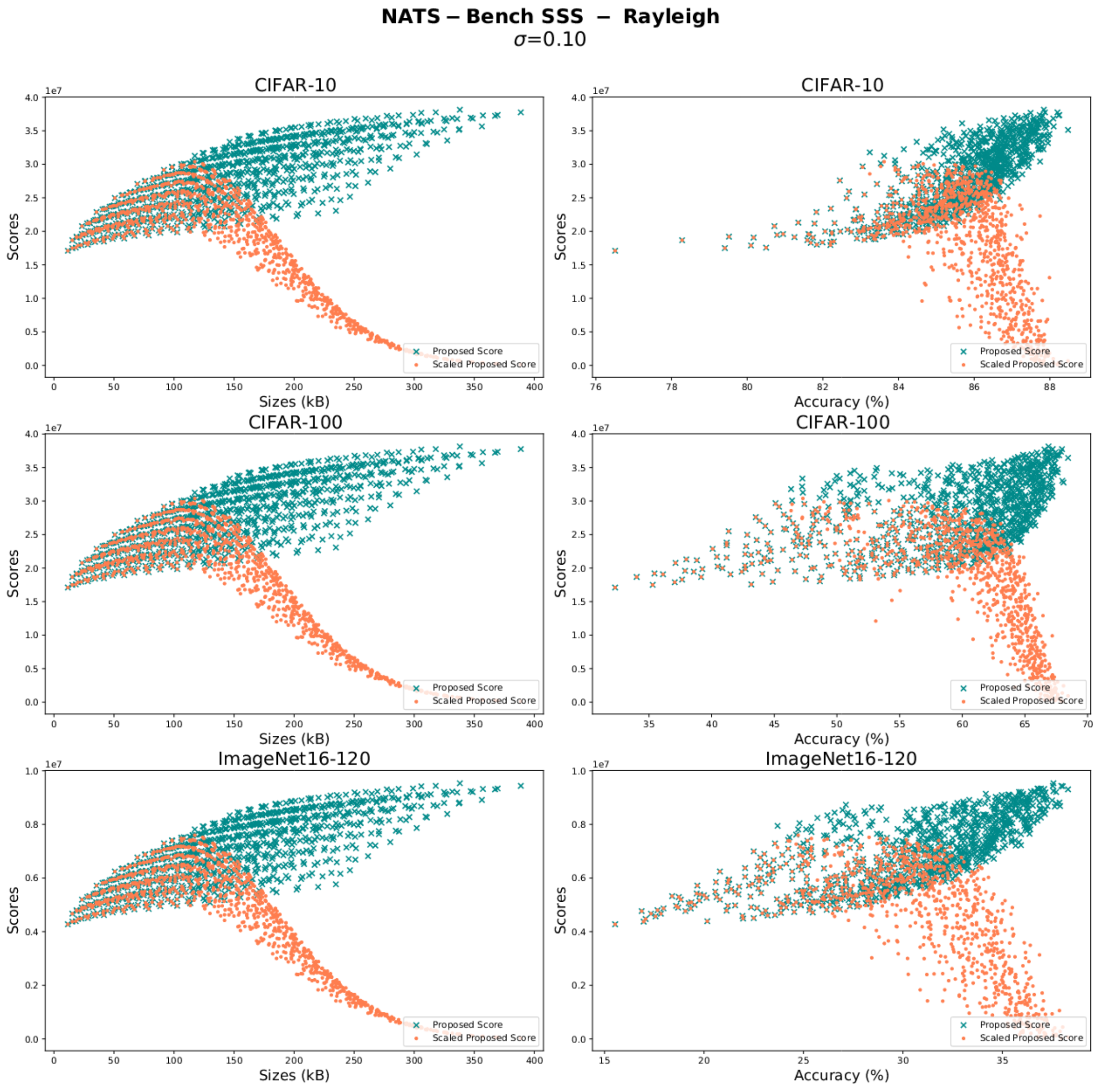
Source: The Author.

Figure 18 – Plots of the proposed metric in NATS-Bench TSS search space with and without scaling using Hyperbolic Secant Distribution as penalty function for network size values superior to 200kB.



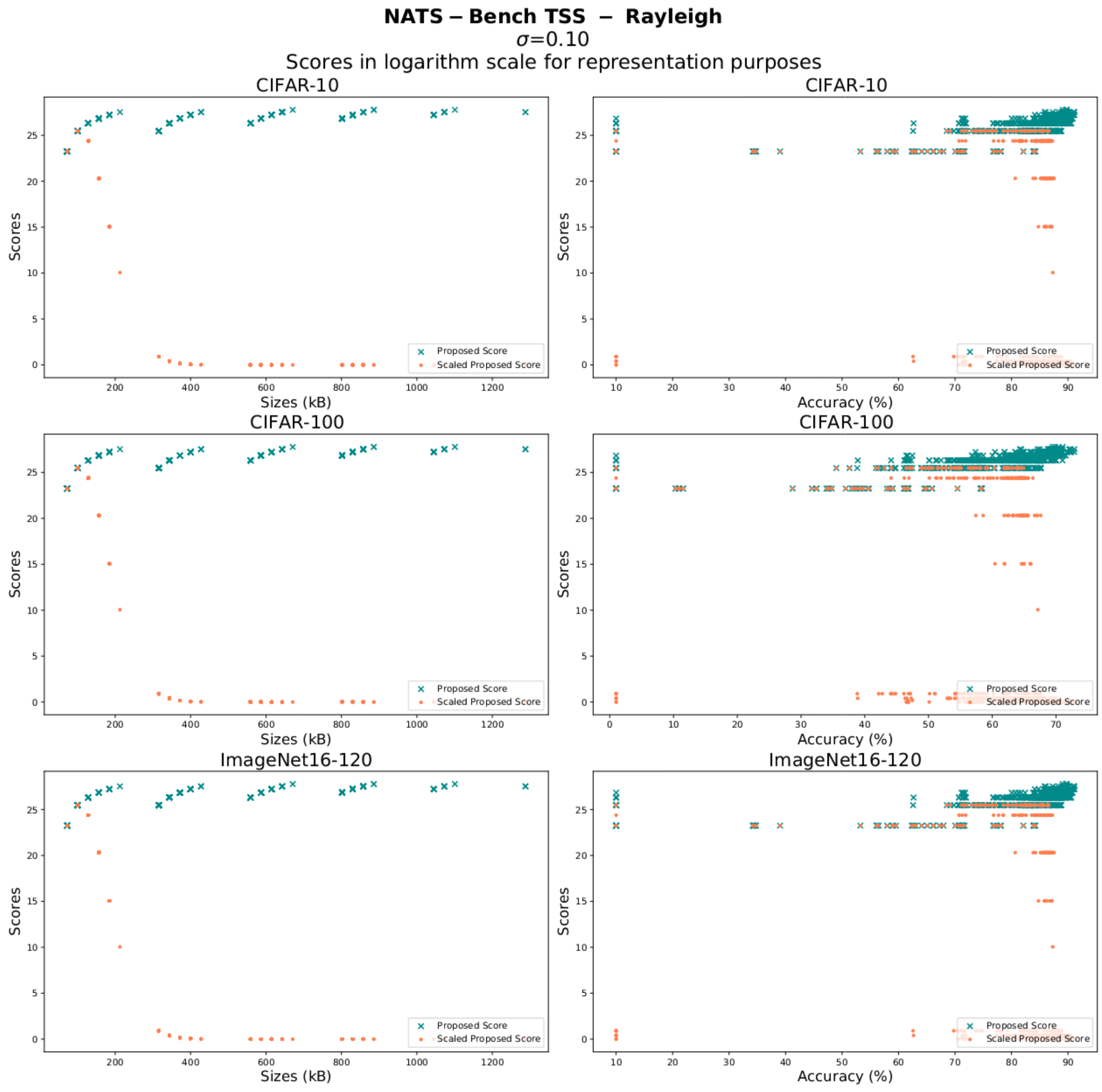
Source: The Author.

Figure 19 – Plots of the proposed metric in NATS-Bench TSS search space with and without scaling using Rayleigh Distribution as penalty function for network size values superior to 200kB.



Source: The Author.

Figure 20 – Plots of the proposed metric in NATS-Bench TSS search space with and without scaling using Rayleigh Distribution as penalty function for network size values superior to 200kB.



Source: The Author.

Table 7 – Results of application of GDF-penalized version of the proposed metric in the context of a practical NAS system. The performance is described in terms of time spent searching, the mean±std. accuracies, and the size in kilobytes, considering $N = 100$ randomly-sampled networks.

Proposed Metric (N=100)					Gaussian Scaled Proposed Metric (N=100)			
Benchmark	**Search (s)**	**validation**	**test**	**size (kB)**	**Search (s)**	**validation**	**test**	**size (kB)**
		CIFAR-10				**CIFAR-10**		
NATS-Bench TSS	29.66	91.0 ± 0.82	87.79 ± 1.01	750.65 ± 20.31	40.6	87.33 ± 2.75	87.89 ± 1.07	327.86 ± 0.019
NATS-Bench SSS	21.86	90.55 ± 1.60	89.62 ± 0.29	305.66 ± 0.096	30.14	87.28 ± 0.20	87.31 ± 0.37	273.64 ± 0.006
		CIFAR-100				**CIFAR-100**		
NATS-Bench TSS	29.66	70.91 ± 1.16	63.03 ± 13.57	543.21 ± 30.80	40.6	65.81 ± 2.2	64.67 ± 7.97	271.74 ± 0.023
NATS-Bench SSS	21.86	67.57 ± 2.34	61.64 ± 3.15	378.90 ± 0.035	30.14	59.01 ± 13.93	59.01 ± 13.93	283.56 ± 0.011
		ImageNet16-120				**ImageNet16-120**		
NATS-Bench TSS	29.66	35.62 ± 16.59	35.86 ± 16.19	343.76± 220.31	40.6	34.69 ± 5.61	32.38 ± 0.72	273.64 ± 0.006
NATS-Bench SSS	21.86	43.02 ± 5.81	42.83 ± 6.38	585.13 ± 89.63	30.14	39.066 ± 1.89	37.13 ± 2.50	248.56 ± 0.009

5 CONCLUSIONS

Neural Architecture Search is an exciting and innovative field of research. The systems that take advantage of NAS are used in many areas, such as image recognition, image classification, hyperparameter optimisation, object detection, etc. In terms of capacity, NAS certainly has enormous potential when it comes to finding the best possible architecture to solve a given problem, independent of how complex it is. However, exploring a lot of solutions with variable complexities is highly expensive in terms of computational resources, making NAS suffer to explore and evaluate entire search spaces. The larger the search space, the more there are proposed architectures to test, train and evaluate. Recent advances in producing NAS benchmarks allowed the researchers to investigate different approaches to avoid certain search costs, giving the opportunity for training-free metrics to emerge.

This work proposed a training-free metric to performance estimation strategy stage, that scores an untrained network at initialization, with a positive correlation to their validation accuracy after training and with the network size. The metric showed the capacity to evaluate a set of a thousand random networks in a matter of seconds by just calculating the maximum number of ReLU activations in the neural network addressed.

Moreover, it is showed that the proposed metric, part of the performance estimation strategy stage, can easily be incorporated into search algorithms, which in this work was proved using the random search. Random search coupled with the proposed metric outperform the amount of time required to evaluate a network of many current training free methods (works like [8], [10] and [9]), for some cases reaching 52.21% of speedup. Conversely, to achieve this time reduction the random search with the proposed metric looses around 3% of the final validation accuracy, as a trade-off. The main idea of the proposed metric would be to fast filtering the worst candidates of a search space, being not the final decider of what is the best neural network possible, but in this case decreasing the time to find the best option available. Therefore, as a filter the proposed metric could be used to list the top N architectures to be trained and evaluated.

In addition, the proposed metric showed a strong correlation with the network size, leading to a entire new proposal of penalizing the metric by its own size. This new approach has great potential to identify good networks with smaller sizes, making possible to improve the speed of systems like μNAS [22] to find a good neural network that can fits an environment with extreme limited resource constrains. The scaling process comes to a 20% time increase, but in advantage is possible to find good networks with half of the size in kilobytes (as it showed by Table 7. Thus, the proposed metric is agnostic to the search algorithm stage, allowing it to be used in any NAS system, making it cheaper to find a powerful and small (in size) architecture.

Finally, it is shown that the proposed metric achieves and overcomes its main objective, by reducing the time evaluating randomly sampled neural networks and providing the penalizing the score by the network's size. These two facts warranty that the proposed metric indeed can be used as a fast filter to discard inaccurate and big networks.

REFERENCES

[1] X. He, K. Zhao, and X. Chu, “Automl: A survey of the state-of-the-art”, *Knowledge-Based Systems*, vol. 212, p. 106 622, 2021.

[2] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning”, *CoRR*, vol. abs/1611.01578, 2016. arXiv: `1611.01578`.

[3] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. V. Le, and A. Kurakin, “Large-scale evolution of image classifiers”, *CoRR*, vol. abs/1703.01041, 2017. arXiv: `1703.01041`.

[4] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey”, *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[5] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning”, *CoRR*, vol. abs/1611.02167, 2016. arXiv: `1611.02167`.

[6] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition”, *CoRR*, vol. abs/1707.07012, 2017. arXiv: `1707.07012`.

[7] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, “Efficient neural architecture search via parameter sharing”, *CoRR*, vol. abs/1802.03268, 2018. arXiv: `1802.03268`.

[8] V. Lopes, S. Alirezazadeh, and L. A. Alexandre, “EPE-NAS: efficient performance estimation without training for neural architecture search”, *CoRR*, vol. abs/2102.08099, 2021. arXiv: `2102.08099`.

[9] J. C. Mellor, J. Turner, A. J. Storkey, and E. J. Crowley, “Neural architecture search without training”, *CoRR*, vol. abs/2006.04647, 2020. arXiv: `2006.04647`.

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

[11] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efficient architecture search”, *CoRR*, vol. abs/1711.00436, 2017. arXiv: `1711.00436`.

[12] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization.”, *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[13] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning”, *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms”, *CoRR*, vol. abs/1707.06347, 2017. arXiv: `1707.06347`.

[15] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies”, *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[16] T. Blickle and L. Thiele, “A comparison of selection schemes used in evolutionary algorithms”, *Evolutionary Computation*, vol. 4, pp. 361–394, 1997.

[17] G. Kyriakides and K. G. Margaritis, “An introduction to neural architecture search for convolutional networks”, *CoRR*, vol. abs/2005.11074, 2020. arXiv: `2005.11074`.

[18] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, “Taking the human out of the loop: A review of bayesian optimization”, *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[19] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, “Neural architecture search with bayesian optimisation and optimal transport”, *CoRR*, vol. abs/1802.07191, 2018. arXiv: `1802.07191`.

[20] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “SMASH: one-shot model architecture search through hypernetworks”, *CoRR*, vol. abs/1708.05344, 2017. arXiv: `1708.05344`.

[21] D. Ha, A. M. Dai, and Q. V. Le, “Hypernetworks”, *CoRR*, vol. abs/1609.09106, 2016. arXiv: `1609.09106`.

[22] E. Liberis, L. Dudziak, and N. D. Lane, “μnas: Constrained neural architecture search for microcontrollers”, *CoRR*, vol. abs/2010.14246, 2020. arXiv: `2010.14246`.

[23] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition”, *CoRR*, vol. abs/1804.03209, 2018. arXiv: `1804.03209`.

[24] X. Dong and Y. Yang, “Nas-bench-201: Extending the scope of reproducible neural architecture search”, in *International Conference on Learning Representations*, 2020.

[25] X. Dong, L. Liu, K. Musial, and B. Gabrys, “Nats-bench: Benchmarking nas algorithms for architecture topology and size”, *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[26] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, “Nas-bench-101: Towards reproducible neural architecture search”, in *International Conference on Machine Learning*, PMLR, 2019, pp. 7105–7114.

[27] I. Radosavovic, J. Johnson, S. Xie, W. Lo, and P. Dollár, “On network design spaces for visual recognition”, *CoRR*, vol. abs/1905.13214, 2019. arXiv: `1905.13214`.

[28] W. Chen, X. Gong, and Z. Wang, “Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective”, *CoRR*, vol. abs/2102.11535, 2021. arXiv: `2102.11535`.

[29] D. Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas - Second Edition*. Princeton University Press, 2009, ISBN: 9781400833344.

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

[31] ——, “Cifar-100 (canadian institute for advanced research)”,

[32] O. Russakovsky, J. Deng, H. Su, *et al.*, “Imagenet large scale visual recognition challenge”, *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[33] L. Li and A. Talwalkar, “Random search and reproducibility for neural architecture search”, *CoRR*, vol. abs/1902.07638, 2019. arXiv: `1902.07638`.

[34] *Google colab pro service*. available from: `https://colab.research.google.com/signup`.