UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Raphael Adamski

Multi-Objective Differentiable Neural Architecture Search
Busca Multiobjetivo e Diferenciável de Arquitetura de Rede Neural

Campinas

2022

UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Raphael Adamski

Multi-Objective Differentiable Neural Architecture Search

Busca Multiobjetivo e Diferenciável de Arquitetura de Rede Neural

Dissertation presented to the School of Electrical and Computer Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master, in Electrical Engineering the area of Computer Engineering.

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Àrea de Engenharia de Computação

Supervisor: Prof. Dr. Fernando José Von Zuben
Co-supervisor: Prof. Dr. Marcos Medeiros Raimundo

ESTE TRABALHO CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO RAPHAEL ADAMSKI, E ORIENTADA PELO PROF. DR. FERNANDO JOSÉ VON ZUBEN

Campinas

2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

Adamski, Raphael, 1993-
Ad196m Multi-objective differentiable neural architecture search / Raphael Adamski. – Campinas, SP : [s.n.], 2022.

Orientador: Fernando José Von Zuben.
Coorientador: Marcos Medeiros Raimundo.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes neurais (Computação). 2. Redes neurais profundas. 3. Otimização multiobjetivo. I. Von Zuben, Fernando José. II. Raimundo, Marcos Medeiros. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Busca multiobjetivo e diferenciável de arquitetura de rede neural
Palavras-chave em inglês:
Neural networks (Computing)
Deep neural networks
Multi-objective optimization
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Fernando José Von Zuben [Orientador]
Gisele Lopo Pappa
Levy Boccato
Data de defesa: 21-03-2022
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-5356-4078
- Currículo Lattes do autor: http://lattes.cnpq.br/6513405765499129

COMISSÃO JULGADORA - DISSERTAÇÂO DE MESTRADO

Candidato(a): Raphael Adamski RA: 227179
Data de defesa: 21 de Março de 2022
Título da Tese: "Multi-Objective Differentiable Neural Architecture Search"

Prof. Dr. Fernando José Von Zuben (Presidente, FEEC/UNICAMP)
Profa. Dra. Gisele Lobo Pappa (Titular externo, DCC/UFMG)
Prof. Dr. Levy Boccato (Titular interno, FEEC/UNICAMP)

A Ata de Defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

Dedico esta dissertação a minha esposa, meus pais e meus familiares que sempre estiveram comigo, me apoiaram e me deram todo suporte para chegar aqui.

Agradecimentos

O presente trabalho foi realizado com apoio do SiDi Campinas que me permitiu estudar 8 horas semanais durante o período de trabalho. Também houve apoio incondicional do Co-Orientador Prof. Dr. Marcos Raimundo em todo o desenvolvimento do projeto. Gostaria de agradecer ao Orientador Prof. Dr. Fernando Von Zuben por me acolher, propor a ideia inicial deste trabalho e me apoiar ao longo de todo o processo de maneira séria, respeitosa e incentivadora. Por fim agradeço à Comissão de Pós-graduação da Faculdade de Engenharia Elétrica e de Computação pela transparência, apoio e agilidade durante todo o período de pós-graduação.

"If I have seen further, it is by standing on the shoulders of giants."
(Isaac Newton in a 1675 letter to Robert Hooke)

Resumo

O tema central desta pesquisa é busca automática de arquiteturas de redes neurais (NAS, do inglês *Neural Architecture Search*), que corresponde ao processo de automação da busca por arquiteturas com mínima interferência humana. O principal objetivo da pesquisa é aplicar técnicas de otimização multiobjetivo (MOO, do inglês *Multi-objective optimization*) para promover diversidade de modelos com diferentes qualidades em NAS com foco em Differentiable ARchiTecture Search (DARTS). Para tanto, foram empregadas as regularizações L_1 e L_2 com *cross-entropy*, capazes de promover redução de complexidade em classificação multiclasse. Ao mesmo tempo, a existência de diversos modelos eficientes aproximando a fronteira de Pareto nos permitiu construir um *ensemble* com o propósito de competir com o estado da arte em NAS. A existência de uma diversidade de modelos permite que o usuário aplique preferências *a posteriori* de interesse prático, selecionando modelos de acordo com sua parcimônia, latência ou acurácia. Os resultados mostraram que efetivamente é possível encontrar modelos otimizados e específicos para cada aplicação, empregando um montante reduzido de recursos computacionais nesta busca e recorrendo a células básicas distintas em sua composição. Por fim, a perspectiva multiobjetivo levou a uma nova abordagem em NAS, a qual explora diretamente a existência de diversidade entre modelos de aprendizado eficientes, caracterizados por compromissos ótimos entre múltiplos objetivos do aprendizado.

Palavras-chaves: Busca de arquiteturas de redes neurais; Otimização multiobjetivo; Comitê de máquinas; Conjunto diverso de modelos eficientes; Busca diferenciável de arquiteturas.

Abstract

The main topic of this research is Neural Architecture Search (NAS), which corresponds to the process of automating architecture engineering with minimal human interference. The goal of the research is to apply Multi-objective optimization (MOO) techniques to promote diversity of models with distinct traits in NAS, with the focus on Differentiable ARchiTecture Search (DARTS). We applied L_1 and L_2 regularization terms with cross-entropy, aimed at promoting the reduction of complexity in multiclass classification. Additionally, the existence of a diverse set of efficient models close to the Pareto frontier is explored to build an ensemble of learning models, with the purpose of achieving the state-of-the-art performance on NAS. This model diversity allows the user to apply *a posteriori* preferences, selecting models according to parsimony, latency or accuracy. The results show that our proposal is capable of finding optimized and specific models for each application with limited resources due to distinct basic cells on their composition. Finally, the explicit multi-objective perspective led to an entirely new approach on NAS which explores directly the existence of diverse efficient models characterized by optimal trade-offs among multiple learning objectives.

Keywords: Neural Architecture Search; Multi-objective optimization; Ensemble learning; Diversity set of efficient models; Differentiable ARchiTecture Search (DARTS).

List of Figures

List of Tables

List of Abbreviations

conv Convolutional layer
batchNorm Batch normalization

acc Accuracy

Valid Validation

crit loss Loss as the optimization criterion
reg loss Loss as the regularization term

params number of parameters

List of Acronyms

NAS Neural Architecture Search

DARTS Differentiable ARchiTecture Search

MOO Multi-objective optimization

PC-DARTS Partially-Connected Differentiable ARchiTecture Search

NISE Non-Inferior Set Estimation

RL Reinforcement Learning

EA Evolutionary Algorithm

FLOPs floating-point operations

SMBO Sequential model-based optimization

NNs Artificial Neural Networks

MLP Multilayer perceptron

GPU Graphics processing unit

TPU Tensor processing unit

CNNs Convolutional Neural Networks

BLO Bilevel optimization

List of Symbols

$\mathcal{L}_{val}$	validation set loss function
$\mathcal{L}_{tr}$	train set loss function
$\mathcal{L}_{reg}$	regularization loss function
L_2	mean square error loss function
L_1	mean absolute error loss function
$\boldsymbol{\alpha}$	architecture operation parameter
$\boldsymbol{\beta}$	architecture concatenation parameter
$\boldsymbol{w}$	neural network model weights
$\boldsymbol{w}^*$	optimized neural network model weights
$\mathbf{w}$	weight vector scalarizing the objective functions
ν	regularization weight scalar
$\mathcal{O}$	space of neural networks operations
o	fixed operation $\in \mathcal{O}$
ξ	learning rate
$\boldsymbol{S}$	binary matrix to mask channels

Contents

1 Introduction

Selecting the learning model and its hyperparameters is crucial to employ adequate machine learning systems. Several challenges are posed in this selection including accuracy, time-response, memory limitations, and fairness. A wide range of methodologies attempt to deal with those challenges, particularly in deep learning. Neural Architecture Search (NAS) and Multi-objective optimization (MOO) are gaining more attention and are already responsible for impressive results, considering multiple performance criteria. NAS selects neural network deep architectures and components with minimal human interference (REN et al., 2021), seeking for the most suitable model giving the constraints (WISTUBA; RAWAT; PEDAPATI, 2019). The associated methods usually rely on evolutionary, gradient-based, Bayesian or reinforcement learning algorithms. On the other hand, multi-objective strategies resort to a pool of learning machines satisfying distinct trade-offs among multiple objectives, namely: high accuracy, low complexity, low hardware latency, high interpretability, high diversity of base cells (JIN; SENDHOFF, 2008). Thus, MOO will explicitly incorporate multiple conflicting objectives on the NAS formulation, potentially yielding more efficient and diverse models (WISTUBA; RAWAT; PEDAPATI, 2019).

NAS helped automatizing the time-consuming, arduous and error-prone task of searching for suitable deep learning architectures, at the cost of an expressive increase in the requirements for computational resources during the search. With that in mind, search methods that are not so computationally intensive are gaining more attention (WISTUBA; RAWAT; PEDAPATI, 2019). Among them, methods employing gradient descent on differentiable deep learning models were faster than their competitors without compromising accuracy improvement. Recent MOO research (RAIMUNDO; DRUMOND, et al., 2021) was able to train multiple learning models with deterministic and automatic spread of efficient learning models along the Pareto frontier (composed of efficient solutions) created by the multinomial loss vs. regularization trade-off. This intersection would result in a huge advantage to the NAS field because high-quality sampling of the Pareto frontier creates a controlled distribution of trade-offs between accuracy and complexity, thus making available multiple and diverse models as well as models with distinct purposes (depending on their complexity).

Although NAS automatically searches for the best deep architectures and reduces the number of parameters to tune, there are still some hyperparameters arbitrarily selected according to the demands of the application, such as the regularization coefficient.

Therefore, we want to explicitly investigate the effect of the regularization coefficient in a multi-objective perspective. Since overall loss (training error) and regularization strength (usually empirically selected) are conflicting optimization objectives, more vigorous regularization can restrain the flexibility of the learning model, potentially increasing the overall loss (RAIMUNDO; DRUMOND, et al., 2021).

Our research then focuses on gradually adding regularization (complexity measure) strength to the model, thus filling an approximation of the Pareto frontier with efficient learning models exhibiting distinct and automatically defined trade-offs between error and model complexity. Subsequently, some *a posteriori* preference of the user may be adopted to select the best deep learning model, with minimum accuracy reduction. The selected NAS implementation, used as a baseline and starting point, was PC-DARTS[1], given the effectiveness of the architecture search and the distinguished performance of the obtained deep learning model. This algorithm has a fixed L_2 regularization by default. In our work, L_2 and L_1 regularization criteria were considered and also a deterministic multi-objective solver called NISE[2] were used to maximally distribute the efficient deep learning models along the Pareto frontier. Notice that more conflicting objectives could have been incorporated in the formulation, such as hardware latency, potentially producing additional diversity in the final deep architectures. However, this extension would require replacing NISE with MONISE (RAIMUNDO; DRUMOND, et al., 2021), an extension of NISE to more than two objectives.

Finally our proposal is able to achieve a diverse set of models that support two major contributions: the possibility of choosing *a posteriori* the best model among a diverse set of distinct trade-offs (in replacement to the proposition of a single option after the NAS execution), and the possibility of implementing a powerful ensemble of efficient and diverse learning models, possibly overcoming the state-of-the-art performance in the literature.

The second and third chapters will depicture the conceptual basis used to support methodology and experiments. On the fourth chapter detailed experiments are proposed and explained to the user. Finally the fifth chapter will analyse the results of the experiments and outline the main contributions.

[1]Partially-Connected Differentiable ARchiTecture Search (XU et al., 2020)
[2]Non-Inferior Set Estimation (COHON; CHURCH; SHEER, 1979)

2 Searching Architectures in NNs

Before introducing the main problem on Section 2.5 we are going to present the general concepts of neural networks, the relevant operations in Convolutional Neural Networks (CNNs) and the motivation for their adoption as the basic architectures on NAS.

2.1 Neural Networks

Mcculloch and Pitts (1943) proposed a mathematical model to represent a neuron. After fifteen years a learnable neuron was first proposed by Rosenblatt (1958), called Perceptron. This is the simplest network, with one unit and adjustable weights, guiding to a trainable linear classifier. Figure 2.1 show this structure and the generic definition is:

$$f(\boldsymbol{w_j}, x) = f(\sum_{i=1}^{m} w_{ji} x_i + w_{j0}) \tag{2.1}$$

where $f(\cdot)$ is the activation function and m is the number of inputs.

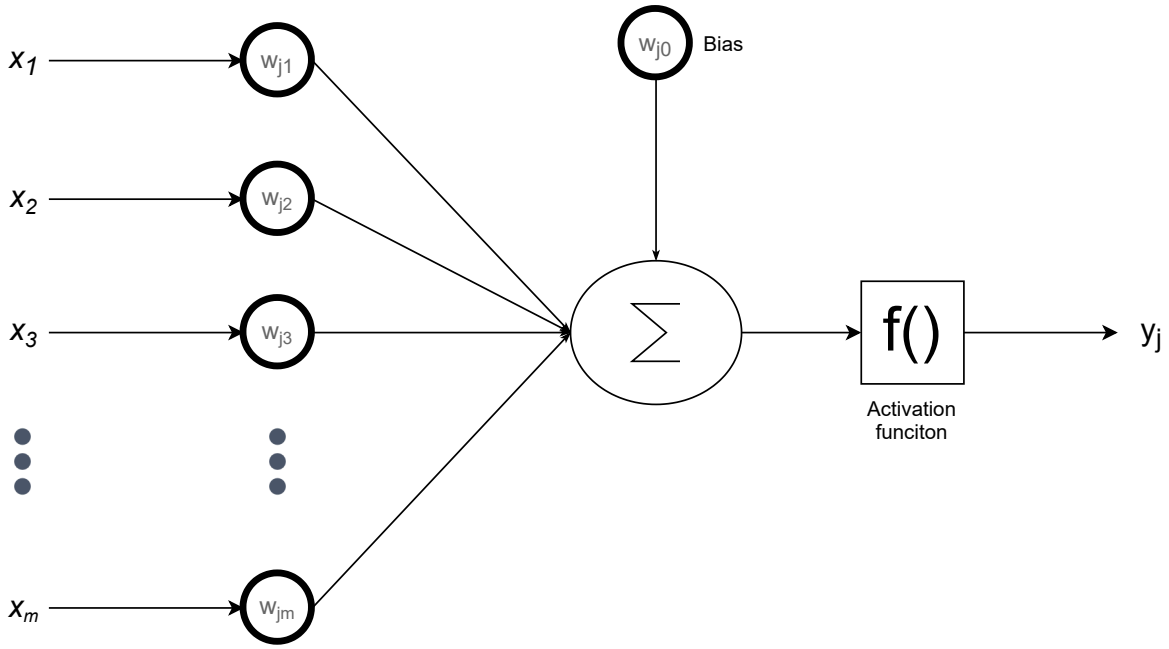


Figure 2.1 – Representation of the Perceptron and its adjustable weights.

Since then, several attempts have been made toward more general neural network models, theoretically capable of universally approximating arbitrary continuous nonlinear functions on a compact input space (HORNIK; STINCHCOMBE; WHITE, 1989). The Multilayer perceptron (MLP) (or more generally speaking **feedforward networks**)

was one of them. It consists of many layers of neurons, each one performing the aforementioned input-output mapping of a single Perceptron. In other words it is a cascade of multidimensional nonlinear mappings. The nonlinear mapping performed by each neuron on the i-th layer with n neurons on the layer before and k neurons on the current layer is given by:

$$o_j^i(x) = f^i(\sum_{a=0}^{n} w_{ja}^i \cdot o_a^{i-1}(x) + w_{j0}^i) \tag{2.2}$$

where $j = [1, ..., k]$, w^i are the weights of the i-th layer and o represents the layer output.

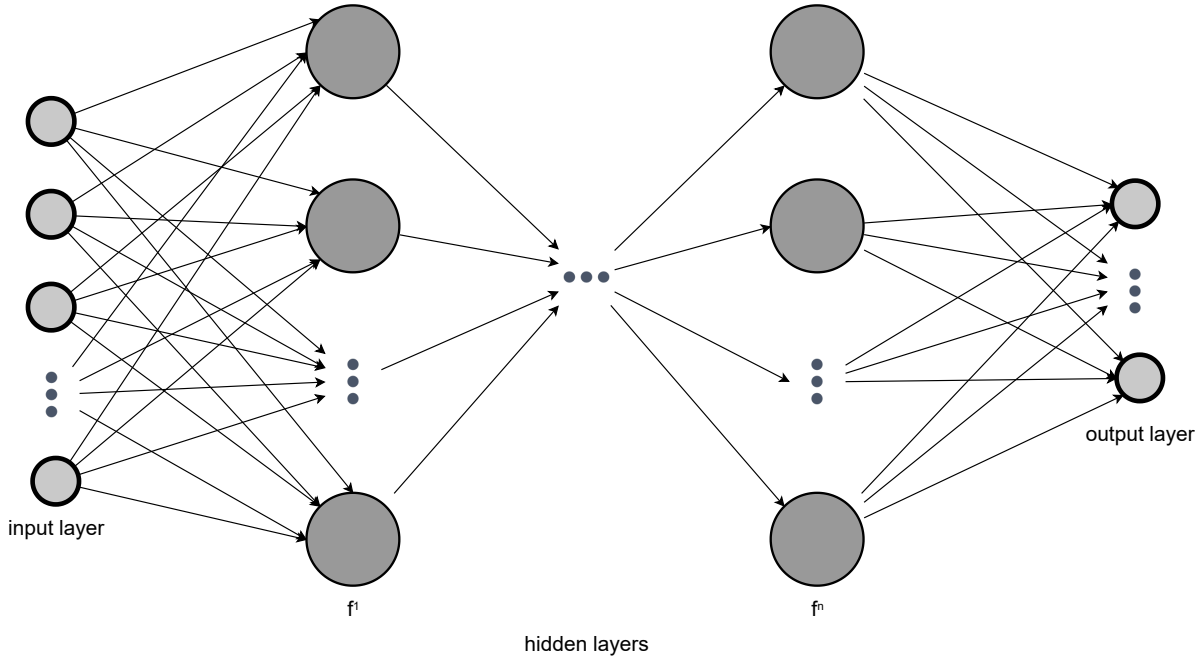


Figure 2.2 – Representation of an MLP. All the outputs of a previous layer are taken as inputs of all the neurons at the subsequent layer, with each input multiplied by an adjustable weight. The dark gray circles represent neurons and the light gray ones represent features from the inputs and output.

This chain of non-linear multidimensional mappings allows multilayer neural networks to solve non-linear problems, by jointly adjusting the whole set of weights with the purpose of reducing the accumulated error produced at the output layer. The output error should then be backpropagated along the layers (RUMELHART; HINTON; WILLIAMS, 1986), thus implementing a learning process founded on gradient descent (WIDROW; HOFF, 1960).

With the advent of cheaper hardwares, faster and dedicated boards (GPUs and TPUs for instance), pretraining and other advanced techniques for making deep learning computationally efficient, it was possible to work with big data and deep architectures in neural network learning. Krizhevsky, Sutskever, and Hinton (2012) took another step with their deep convolutional networks, in which the convolutional layers were further

demonstrated to perform representation learning (BENGIO; COURVILLE; VINCENT, 2013) by extracting along the layers a hierarchy of relevant features from the input data.

2.2 Operations in Convolutional Neural Networks

Convolutional Neural Networks (CNNs) played a decisive role on the recent attention at deep learning. It happens due to the feature extraction ability of the convolutional operator (MARQUES, 2018), which consists of a small kernel convoluted (or cross-correlationed) throughout a whole input image with a specific stride and padding. This creates a sparse interaction meaning that we need to store fewer parameters compared to a fully connected layer (see Figure 2.2) (GOODFELLOW; BENGIO; COURVILLE, 2016).

The CNNs, on the other hand, exhibit a dimensionality issue along the cascade of convolutional layers. For instance, a color input image of $32 \times 32 \times 3$ when processed by a Convolutional layer (*conv*) with 16 kernels of 3×3 (see Fig. 2.3) yields a $30 \times 30 \times 16$ output. Aiming at reducing the amount of weights for the subsequent convolutional layer, a pooling operator can be introduced between consecutive *conv*s.

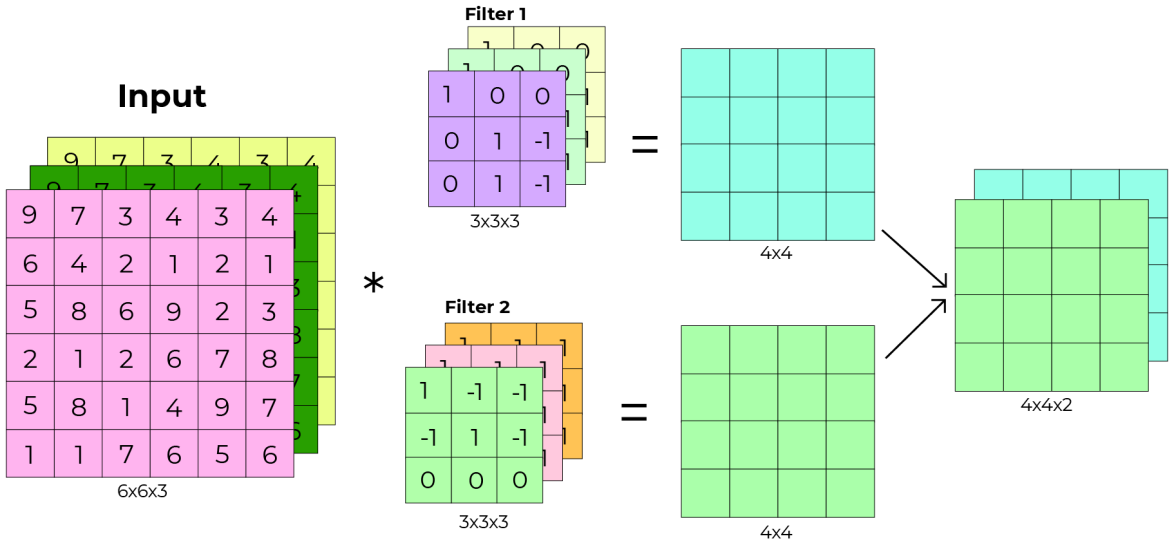


Figure 2.3 – Representation of a 2D Convolution with 2 kernels (channels or filters), height 3, width 3, stride 1, and 0 padding.

The pooling operator is applied normally on the height and width of the image, leaving the channel dimension intact (MARQUES, 2018). The most known and used pooling operators are the **max pooling** and the **(weighted) average pooling** shown on Fig. 2.4. The total number of weights in CNNs may also be reduced by considering separable convolution operators (GOODFELLOW; BENGIO; COURVILLE, 2016).

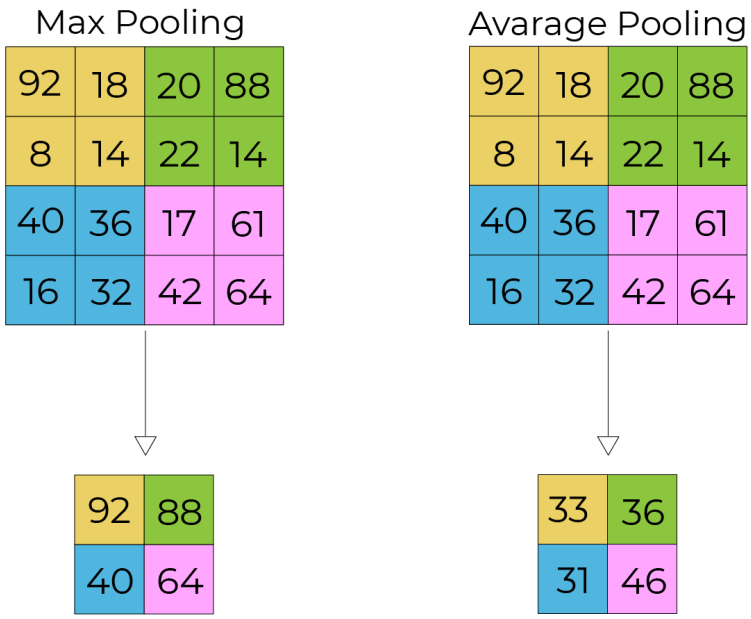


Figure 2.4 – Representation of pooling operation on one dimension with height 2, width 2, stride 2, and 0 padding.

Other challenges for the computational efficiency of the training process are verified when the neural architecture becomes deeper, such as the vanishing and the exploding gradient due to the increasing number of layers that can make the gradient near zero on the first stages of backpropagation algorithm, or even accumulate the residuals until it explodes. This was addressed using many techniques: smaller **learning rate** (along with slower training), proper **weight initialization** (GLOROT; BENGIO, 2010), non-saturating **activation functions** (NAIR; HINTON, 2010), **gradient clipping** (PASCANU; MIKOLOV; BENGIO, 2013), **skip-connection** (HE, K. et al., 2016) and **Batch normalization (*batchNorm*)** (IOFFE; SZEGEDY, 2015).

Differently from other mentioned techniques, **Batch normalization** is a layer added to the model (to the output of any input or hidden layer) with adaptive reparametrization (GOODFELLOW; BENGIO; COURVILLE, 2016) and is explored on our baselines. The normalization is applied to a distinct mini-batch at each step of a given training epoch. Let X be a minibatch of outputs from a layer where each row i is a sample to normalize:

$$X' = \frac{X - \mu}{\sigma} \tag{2.3}$$

where μ is the mean and σ is the standard deviation, both calculated independently for each feature of the mini-batch. For the i-th sample point, the update would be $x'_i = batchNorm(x_i, X)$. This way it propagates (forward and backward) values unaffected by the scale of the layer parameters, stabilizing the growth of the parameters and allowing higher learning rates (consequently faster training).

The **skip-connection** (shortcut connection or identity function) is not a layer by definition, but it creates a direct connection between two non-consecutive layers, without additional weights. Kaiming He et al. (2016) successfully used this type of operation with stacked blocks (set of predefined subnetworks), very similar to the architecture of our baselines. This connection helps the lower layers to receive a gradient that is not vanishing or exploding, mitigating the training degradation and then promoting accuracy improvement on deeper networks (HE, K. et al., 2016).

2.3 Hyperparameter Optimization

Despite the increasing accessibility to NNs, the number of design decisions to make, involving both architecture definition and weight adjustment procedure, tends to increase uncontrollably, making it impossible to test all architectures and all aspects of the optimization algorithm in the search of the best model for each problem.

Those decisions that control the behavior of NNs when solving learning problems (GOODFELLOW; BENGIO; COURVILLE, 2016) are called hyperparameter because they are decided before the training and are not learned. Some relevant examples are: number of layers; type of layers (operations described on Sec. 2.2); number of neural units (e.g. number of convolutional kernels) at each layer; learning rate; convolution kernel size; loss metric; and the optimization algorithm.

Some techniques were develop to avoid manual testing, like grid search, Bayesian optimization or evolutionary optimization. Although powerful for continuous variables, like learning rate and weight decay, they struggle with the discrete parameters that are not easy to differentiate. In this context NAS was able to improve architecture design and show impressive results.

2.4 Neural Architecture Search (NAS)

NAS aims at automatically finding the best architectures given a data set, thus avoiding human hyperparameter intervention. This search depends on the **search space** (modular, or global) and the **optimization method** (how to generate new candidate solutions and test them), necessarily relying on some human expertise to tune the ultra-parameters that will determine how to search the space of network hyperparameters.

2.4.1 Search Space

We can classify the search space into two categories: (1) Global/Macro space, which defines graphs that represent the entire network. (2) Cell-based space, which defines

a block or basis subnetwork that will be cascaded to compose the whole network.

Global Search allows more freedom and possibilities for the network. Each iteration of this method combines sequence of nodes and trains a model using this structure. The order, quantity, operations and hyperparameters inside each node enables flexibility that allow an explosive amount of combinations, thus imposing a challenge since it is impossible to test all. Baker et al. (2016) started pruning notably poor or expensive architectures. Further works added more freedom with skip-connections (ZOPH; LE, 2017), but had to reduce the search space toward feasibility.

Cell Search is based on rewarded handcrafted networks with many repeated basic structures like fully-connected or convolutional layers (here called cells). The goal is to find one normal cell (responsible for finding the best operations that yield most loss function reduction) and one reduction per search (responsible for reducing spatial resolution) that will become the basic units of the final network, thus reducing the number of possibilities in the search. Each cell is composed of nodes with a fixed topology but various operations connecting those nodes. Those different combinations of order and layers create the population of cells to test. Zoph, Vasudevan, et al. (2018) proposed a reduction cell to handle the feature dimension and most following works also used it. Normal cells use stride one to avoid changing the dimensions and the most common number of nodes is 5. Reduction cells use stride two to reduce spacial size on intermediate points. Each block can have multiple inputs, one or two of the previous cells, then creating skip-connections. Cell-based search is usually composed of two steps (See Subsection 2.4.3), one devoted to search the cell and another to train the final net from scratch using the obtained cells. Relevant studies (CAI; ZHU; HAN, 2019; CAI; GAN, et al., 2020) tried to avoid surrogate models like this, but, nonetheless, proxies improve the speed and exhibit accuracy very close to the ones achieved by non-proxy algorithms.

2.4.2 Optimization Methods

In this section we will discuss different optimization algorithms to search the best architecture $\boldsymbol{\alpha}$ with respect to the objective function (e.g., minimizing the loss function taken as accuracy or cross-entropy). The first high quality NAS algorithms used Reinforcement Learning (RL) (ZOPH; LE, 2017; BAKER et al., 2016) and neuroevolution (REAL; MOORE, et al., 2017) consuming hundreds of GPU hours to find appropriate architectures (WISTUBA; RAWAT; PEDAPATI, 2019).

Reinforcement Learning models an agent that takes decisions to change the environment, i.e the architecture, to maximize its reward, usually accuracy, taking into account the past state-action moves. Although most cases limits the iteration to finite states and

episodes, those algorithms were mainly used in the early achievements of the field (ZOPH; LE, 2017), being characterized by a computational overhead. This issue is presented at Table 2.1 where the *GPU Days* column reveals a high computational burden.

Evolutionary Algorithm relies on a population of candidate architectures for NAS that suffer mutation (recombination does not show good results in this case) throughout the epochs to generate new models based on the fitness results. The mutations can occur in a cell or global space (definition made in Section 2.4.1). The most relevant proposals are founded on genetic algorithms. Due to the mutation step of the algorithm, it is very expensive training the networks from scratch after each iteration. The first relevant work from Real, Moore, et al. (2017) using global search space was too permissive allowing redundant operations. Later on Real, Aggarwal, et al. (2018) the notable AmoebaNet-B and AmoebaNet-C strategies were proposed and set the new records on image classification task dataset CIFAR-10 and ImageNet at the cost of 3,150 GPU hours.

One-Shot Architectures search method consists of training only one deep architecture during the search process. This drastically reduces the search time cost but increases the memory cost because it requires an over-parameterized network to be trained. This technique is very suitable to work with Surrogate models, Cell Search and gradient descent optimizers. Aiming at training a single super-architecture, researches used many interesting ideas: in Zoph and Le (2017) an RL controller samples the architecture and then, using gradient descent, updates the architecture and the controller weights. The network parameters are updated on another step. Evidently, this causes optimization issues due to constant structural changes in the neural architecture which does not share weights smoothly. Liu, Simonyan, and Yang (2019) tried to train the entire network with all possibilities at once. They proposed a linear combination of the operations on a cell-style network using softmax, allowing them to make the architecture differentiable. The parameters of the softmax operation (minimized by loss) highlight the best operation to use for the final training step.

Random Search has proven to be an extremely strong baseline method when the search space is known to sample well-performing architectures (WISTUBA; RAWAT; PEDAPATI, 2019). Yu et al. (2019) argued that the search methods still deliver good results due to the very limited search space. They provide empirical evidence that a random search outperforms many of the previously described methods in the search for an RNN cell. Li and Talwalkar (2019) also confirm this result and additionally show that random search finds architectures that perform at least as well as the ones obtained from established optimizers for CNNs.

2.4.3 Surrogate models

A surrogate model is indicated when an outcome of interest cannot be easily obtained, and a function approximation of the outcome is then adopted (WISTUBA; RAWAT; PEDAPATI, 2019). In the case of very costly calculations, a surrogate model may represent an interesting efficiency vs. precision trade-off. In the case of NAS this technique avoids training the entire network while testing different setups. A surrogate model is similar to the original one but it can be smaller (also called proxy model) or trained for fewer epochs (early stopping) and the accuracy is not required to be high because there will be another step to train the final net, thus alleviating the load during search iterations. Given that, the ranking for architectures is desired as long as it is useful, otherwise it can misguide the decision and invalidate the process.

Table 2.1 – Networks performance comparison on CIFAR-10 (NAS and human-designed) most relevant results. Networks were organized by optimization method: Reinforcement Learning (RL), Evolutionary Algorithm (EA), One-shot (gradient optimization) and random search. *GPU Days* refers to total search time of each model and it can be calculated by $GPU\ Days = \#\ of\ GPUs \times t$ where t is the number of days the process took to run (LIU, Y. et al., 2021). Data was extracted from Ren et al. (2021).

Search Method	Research	Error Acc (%)	Params (M)	*GPU Days*
Human-designed	PyramidSepDrop + ShakeDrop	2.67	26.2	N/A
	Shark	3.55	2.9	
	ResNet	6.41	1.7	
RL	ProxylessNAS-R*	2.3	5.8	8.3
	Path-level EAS*	2.49	5.7	200
	NASNet-A*	2.65	3.3	2,000
EA	AmoebaNet	3.34	3.2	3,150
	Neuro-Cell-based Evolution*	3.57	5.8	0.5
	Hierarchical-EAS	3.75	15.7	300
One-Shot	ProxylessNAS-G*	2.08	5.7	8.3
	P-DARTS*	2.50	3.4	0.3
	PC-DARTS*	2.57	3.6	0.1
	SGAS	2.66	3.7	0.25
	GDAS-NSAS	2.73	3.54	0.4
	DARTS (1^{nd} order)*	3.00	3.3	1.5
	DARTS (2^{nd} order)*	2.76	3.3	4
Random	RandomNAS-NSAS	2.64	3.08	0.7
	RandomNAS*	2.85	4.3	2.7
	DARTS Random	3.29	3.2	4

* results using Cutout (DEVRIES; TAYLOR, 2017), a data augmentation technique.

2.5 Differentiable Neural Architecture Search

During the search for an architecture, it was usually assumed that each component or operation should be present or not in a discrete manner. However, Liu, Simonyan, and Yang (2019) surpassed this limitation by obtaining a differentiable architecture with a linearly weighted combination of operations on a surrogate model (WISTUBA; RAWAT; PEDAPATI, 2019). After training architecture and operations weights interchangeably, the final architecture is chosen based on the largest weights of the linear combination. The following subsections will explain this proposal in details and the main recent advances.

2.5.1 Problem Characterization

NAS can be described as a two-part problem: find the best weights given the best architecture. Previous initiatives to solve this optimization problem guided to laborious job that required testing as many architectures as possible. Based on a gradient-based formulation, the whole optimization problem can be described as a Bilevel optimization (BLO) problem:

$$\min_{\boldsymbol{\alpha}} \quad \mathcal{L}_{val}(\boldsymbol{w}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}) \tag{2.4}$$

$$s.t. \quad \boldsymbol{w}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{w}} \mathcal{L}_{tr}(\boldsymbol{w}, \boldsymbol{\alpha}) \tag{2.5}$$

where $\mathcal{L}_{val}$ and $\mathcal{L}_{tr}$ refer to loss value on validation and training sets, respectively, $\boldsymbol{w}$ represents the vector of network weights and $\boldsymbol{\alpha}$ stands for the architecture weights (HE, C. et al., 2020). In this case Eq. (2.4) represents the outer (or upper level) problem, which searches the best architecture $\boldsymbol{\alpha}$ given the optimized weights $\boldsymbol{w}^*$ from Eq. (2.5), which represents the inner (or lower level) problem (LIU, R. et al., 2021) that optimizes the weight $\boldsymbol{w}$ given a fixed architecture $\boldsymbol{\alpha}$.

This problem requires that the network be entirely trained for each $\boldsymbol{\alpha}$ update. This would be prohibitive and further research proposed heuristics to approximate $\boldsymbol{w}^*(\boldsymbol{\alpha})$ or even the adoption of a single level optimization method where the current $\boldsymbol{w}$ is $\boldsymbol{w}^*(\boldsymbol{\alpha})$, resuming the problem to:

$$\min_{\boldsymbol{w},\boldsymbol{\alpha}} \mathcal{L}_{tr}(\boldsymbol{w}, \boldsymbol{\alpha}) \tag{2.6}$$

speeding up the process but with a tendency to degrade the performance (LIU; SIMONYAN; YANG, 2019).

2.5.2 Related work

Having in mind that a multi-objective perspective of NAS would require iterating on the NAS process many times, we started excluding evolutionary, RL and some

gradient-based algorithms that have high GPU footprint. Hence we selected an extension of Liu, Simonyan, and Yang (2019) work (PC-DARTS) as the base NAS strategy for our research due to its good results, low computational burden for the search process (existence of proxy networks) and availability of open source code.

Differentiable ARchiTecture Search (DARTS) (LIU; SIMONYAN; YANG, 2019) was the first NAS gradient-based algorithm with low cost and good results. It changed the field showing that the architecture could be tuned using gradient descent with all candidate operations in a One-Shot architecture. Many subsequent researches tried to improve its speed, results and reduce the proxy model (Search stage) and final model (Evaluation stage) (XU et al., 2020; CHEN et al., 2019; JIANG et al., 2019; GREEN et al., 2020).

The search step creates a network with L sequential cells. Each cell has N nodes, where each node defines a network layer with multiple concurrent operations. Each node N is connected to all previous nodes. There is a pre-defined space of operations denoted by $\mathcal{O}$, in which each element, $o(\cdot)$, is a fixed operation (e.g., identity connection, and 3×3 convolution) on a network layer. Within a cell, the goal is to choose one operation from $\mathcal{O}$ to connect each pair of nodes. Let a pair of nodes be (i, j), where $0 \leq i < j \leq N-1$, the core idea of DARTS is to formulate the information propagated from node i to j as a weighted sum over $|\mathcal{O}|$ operations:

$$f_{i,j}(x_i) = \sum_{o\in\mathcal{O}} \frac{exp\{\boldsymbol{\alpha}^o_{i,j}\}}{\sum_{\theta\in\mathcal{O}} exp\{\boldsymbol{\alpha}^\theta_{i,j}\}} \cdot o(x_i) \tag{2.7}$$

where x_i is the output of the node i and $\boldsymbol{\alpha}$ is the weight for operation $o(x_i)$. The output of a node is the sum of all node inputs transformed $\sum_{i<j} f_{i,j}(x_i)$. The output for the cell is the sum of all nodes that are not cell‘s input. There is an overview at Figure 2.5.

The search finds two types of cells: (i) the **normal cell** which preserves the spatial dimensions and has stride 1 on its operations when applicable and (ii) the **reduction cell**, in which all the operations adjacent to the input nodes are of stride two. The architecture encoding therefore is ($\boldsymbol{\alpha}_{normal}$, $\boldsymbol{\alpha}_{reduce}$), where $\boldsymbol{\alpha}_{normal}$ is shared by all the normal cells and $\boldsymbol{\alpha}_{reduce}$ is shared by all the reduction cells (LIU; SIMONYAN; YANG, 2019).

Half of training samples are used to train the architecture parameters $\boldsymbol{\alpha}$ and the other half to train the operations weights. To speed-up the process, two types of approximation are proposed:

First-order Approximation (Eq. (2.6)**):**

$$\boldsymbol{w}^*(\boldsymbol{\alpha}) \approx \boldsymbol{w}$$

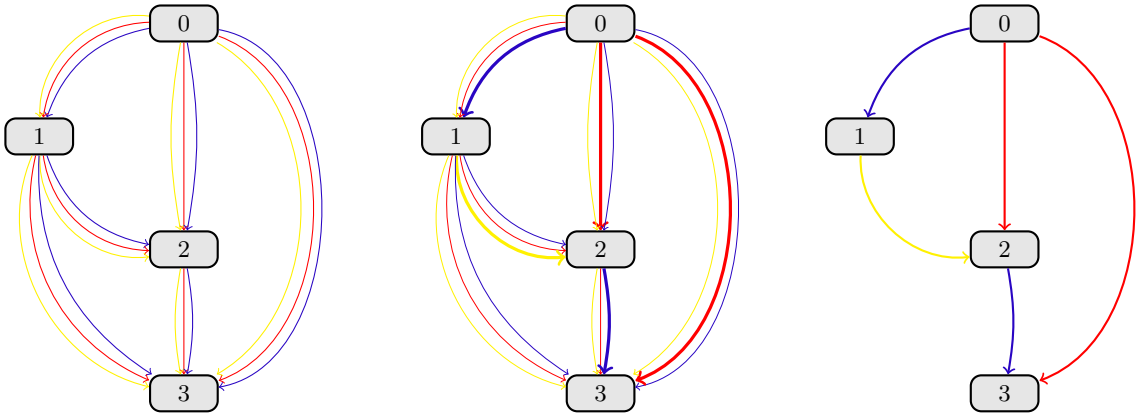


Figure 2.5 – DARTS overview at the cell level of DARTS search: (a) Initially it has the same probability for all operations. (b) Different operations start to be highlighted given the optimization algorithm. (c) At the end of Search stage, the most likely operation creates the discrete architecture cell. Please be aware that *no connection* is an operation under evaluation and it can be selected, which means removing the connection itself from the cell. This image was based on Liu, Simonyan, and Yang (2019) Figure 1.

Second-order Approximation:

$$\boldsymbol{w}^*(\boldsymbol{\alpha}) \approx \boldsymbol{w} - \xi \nabla_w \mathcal{L}_{tr}(\boldsymbol{w}, \boldsymbol{\alpha})$$

where ξ is the current learning rate.

This allows DARTS to train the architecture (BLO upper level - Eq. (2.4)) and weights (BLO lower level - Eq. (2.5)) interchangeably. Each epoch consists of an update of $\boldsymbol{\alpha}$ followed by an update of weights. Although they did not provide convergence guarantees, the algorithm was able to reach a fixed point (LIU; SIMONYAN; YANG, 2019). Finally the second-order approximation was considered better empirically (see Table 2.1).

Afterward, only the operation with the biggest $\boldsymbol{\alpha}^o_{i,j}$ ($f_{i,j} = \arg\max_{o \in \mathcal{O}} \boldsymbol{\alpha}^o_{i,j}$) for each node is maintained in the final discrete network. It can be visualized in Figure 5.4 and it is suitably encoded in a string that we will call architecture code from now on. Then the Evaluation stage is done sequentially connecting J cells, normally bigger then L, and training for more epochs. The number J can vary depending on the dataset, DARTS used $L = 8$ and $J = 20$ for CIFAR-10 and $J = 14$ for ImageNet (mobile settings). Since DARTS rely on Surrogate models ranking and the datasets are similar, they empirically claim that the cell is transferable to ImageNet dataset, avoiding another searching process.

Partially-Connected Differentiable ARchiTecture Search (PC-DARTS) is an extension that samples a subset of the network on Search stage to reduce the redundancy

in exploring the network space. It helps with the high memory footprint on GPUs (that are not expandable memory normally) allowing to train with a larger batch size, yielding faster and better generalization due to stability when back propagating the gradient (XU et al., 2020).

The innovation is the use of partial connection on the channel dimension of the operations. $1/K$ proportion of the channels are calculated on the operation between edges, while $1 - 1/K$ proportion is bypassed and concatenated with the operation output as follows:

$$f_{i,j}^{PC}(x_i; \boldsymbol{S}_{i,j}) = \sum_{o \in \mathcal{O}} \frac{exp\{\boldsymbol{\alpha}_{i,j}^{o}\}}{\sum_{\theta \in \mathcal{O}} exp\{\boldsymbol{\alpha}_{i,j}^{\theta}\}} \cdot o(\boldsymbol{S}_{i,j} * x_i) + (1 - \boldsymbol{S}_{i,j}) * x_i \tag{2.8}$$

where $S_{i,j}$ assigns 1 to selected channels and 0 to masked ones. By varying K, we could trade off between architecture search accuracy and efficiency. For instance, DARTS is equivalent to $K = 1$.

Edge Normalization

Sampling channels reduces the difference between two sets of hyperparameters $\boldsymbol{\alpha}_{i,j}^{o}$ when optimizing the architecture, but operations with weights would propagate inconsistent information across iterations before their weights are well optimized, thus creating a biased preference for a weight-free operation (skip-connection, max pooling, etc.) over a weight-equipped one (convolutions) in $\mathcal{O}$.

To mitigate the fluctuation of the channel sampling on the architecture parameters $\boldsymbol{\alpha}_{i,j}^{o}$, a new set of parameters was introduced after the operation output using the same softmax logic to select edges. Instead of concatenating all the operations output, it was introduced a new parameter $\boldsymbol{\beta}_{i,j}$ between edges (i, j):

$$x_j^{PC} = \sum_{i<j} \frac{exp\{\boldsymbol{\beta}_{i,j}\}}{\sum_{i'<j} exp\{\boldsymbol{\beta}_{i',j}\}} \cdot f_{i,j}^{PC}(x_i) \tag{2.9}$$

Figure 2.6 helps to visualize this new parameter. The final edge (i, j) weight given by the network optimization is obtained multiplying $\boldsymbol{\alpha}_{i,j}$ softmax value by $\boldsymbol{\beta}_{i,j}$ softmax value.

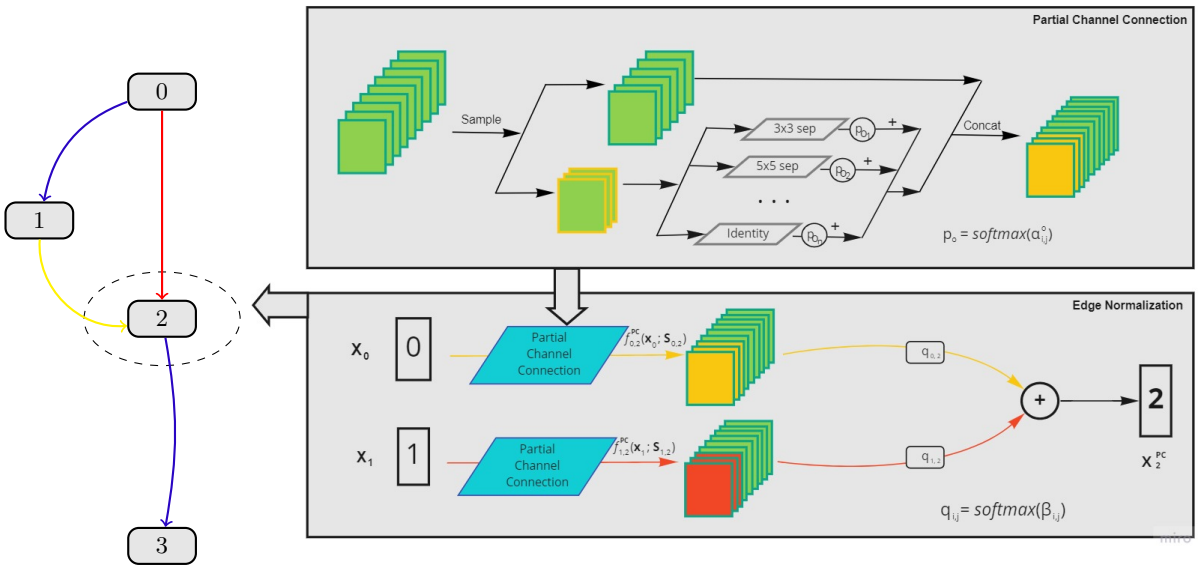


Figure 2.6 – PC-DARTS overview. $\boldsymbol{\alpha}^o_{i,j}$ (operation parameter) and $\boldsymbol{\beta}_{i,j}$ (edge parameter) represent the architecture parameters between nodes, where $0 \leq i < j$ and $o \in \mathcal{O}$. Only a subset, 1/K, of channels are used to update $\boldsymbol{\alpha}^o_{i,j}$ so that the memory consumption is reduced by K times. This image was based on Figure 1 in Xu et al. (2020).

3 Multi-objective optimization (MOO)

MOO is the field of Mathematical Programming that studies optimization problems with more than one conflicting objectives (MIETTINEN, 2012). The conflict implies that the multiple objectives cannot be simultaneously optimized, thus imposing a trade-off among the active objectives. Hence, there is no single best solution, but many solutions with different trade-offs for all objectives, denoted Pareto-optimal (or efficient) solutions. Relevant multi-objective techniques focus primarily on finding a good representation of this set of efficient solutions (RAIMUNDO; DRUMOND, et al., 2021).

Hyperparameter tuning methods are often guided solely by the performance on the validation set, ignoring eventual intrinsic properties of the trade-offs between the different parameters considered. Sampling and populating the Pareto frontier is an interesting way to do so, because it produces a population of diverse efficient solutions (or approximated efficient solutions if the Pareto frontier could not be sampled exactly), avoids the waste of resources, supports model selection and provides a pool of diverse components to feed ensembles (RAIMUNDO; DRUMOND, et al., 2021). Moreover, works in the field rarely take advantage of deterministic multi-objective optimization methods, which are particularly useful for performing a controlled distribution of efficient solutions, properly reacting to the particular conformation of the Pareto frontier. Raimundo, Drumond, et al. (2021) illustrated this advantage on his research using the NISE algorithm (see Figure 3.1).

Formally a multi-objective problem can be defined as (based on (RAIMUNDO, 2018)):

$$\begin{aligned} \min_{x} \quad & \mathbf{f}(\mathbf{x}) \equiv \{f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})\} \\ s.t. \quad & \mathbf{x} \in \Omega, \Omega \subset \mathbb{R}^n \\ & \mathbf{f}(\cdot) : \Omega \to \Psi, \Psi \subset \mathbb{R}^m \end{aligned} \tag{3.1}$$

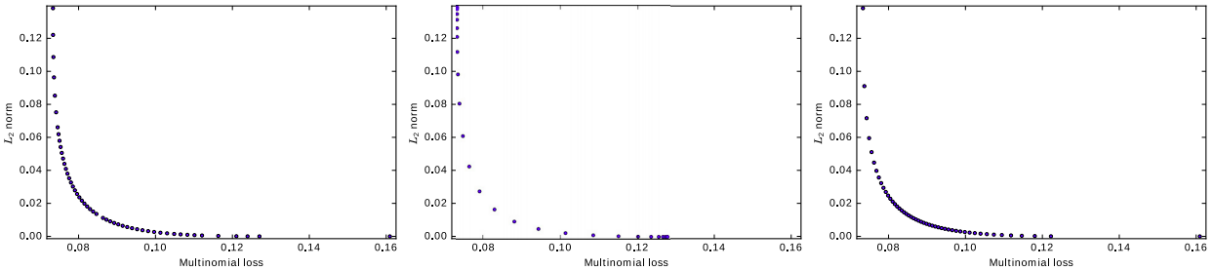
where the set $\Omega \subset \mathbb{R}^n$ is known as the decision space and $\Psi \subset \mathbb{R}^m$ is known as the objective space. The objective function maps values from the decision space to the correspondent point at the objective space.

Since the objective space is multidimensional, two solutions only have a dominance relation when the worse solution (dominated solution) has, with respect to a better solution (non-dominated solution), all objectives of equal or lower quality and at least one objective strictly of lower quality (WIECEK; EHRGOTT; ENGAU, 2016). The solutions not dominated by any other feasible solution are called Pareto-optimal solutions (also called efficient solution, further defined). Given so, it is also relevant to define two other

concepts:

- Efficiency/Pareto-optimality: A solution $\mathbf{x}^* \in \Omega$ is efficient (Pareto-optimal) if there is no other solution $\mathbf{x} \in \Omega$ such that $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*), \forall i \in \{1, 2, ..., m\}$ and $f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$ for some $i \in \{1, 2, ..., m\}$.
- Efficient front/Pareto frontier: An efficient front Ψ^* (Pareto frontier) is the set of all efficient solutions. When considered the problem on Eq. (3.1), the efficient front Ψ^* is formed by efficient objective vectors $\mathbf{f}(\mathbf{x}^*) \in \Psi^*$ which has a corresponding feasible solution $\mathbf{x}^* \in \Omega$.

Beyond single model selection, multiple models can be combined to obtain a committee machine or ensemble. Generally, the synthesis of an ensemble involves three steps: generation of learning machines, selection of a proper subset of these machines, and the composition of the selected machines to achieve a single outcome (ZHOU, 2012). Multi-objective approaches usually address multiple performance metrics (instead of solely model losses or regularization strengths) in the first two steps of the ensemble framework.



(a) NISE (b) Log grid search (c) Constant grid search

Figure 3.1 – Pareto frontier representation for the heart-cleveland dataset. Each blue point corresponds to an optimal value of the regularized multinomial regression with distinct Multinomial loss vs L_2 norm trade-off.
Extracted with permission from Raimundo (2018) Figure 20.

3.1 Multi-objective NAS

Although convex objective functions favor the sampling of a convex Pareto frontier, there is no guarantee it will happen in a multi-objective deep learning problem due to the following reasons: (i) the stochastic initialization, which promotes a distinct outcome at each execution; (ii) it generally has a non-convex loss function (guiding to the existence of local minima); (iii) the nondeterministic behaviour of the optimization given the precision rounding and CUDA convolutions on GPUs; (iv) the usage of proxy networks that are not fully trained. All those reasons can generate different architectures,

loss and accuracy for very similar settings. Furthermore the combination of operations can be sometimes harmful for the network, just like excessive or restrained regularization, making even harder to have a well-behaved function.

Nevertheless the early NAS formulations that yield good results focused mainly on showing the relevance and promisingness of the field. Just after the first results, researches started to care about other factors such as number of parameters, latency and models for specific hardwares. The new objectives could impose new constraints or penalties or be considered as the *a posteriori* preference of the decision maker.

Previous approaches using MOO on NAS are not computationally viable by focusing on an evolutionary search engine (KIM et al., 2017; ELSKEN; METZEN; HUTTER, 2019):

Hsu et al. (2018) tried to use a weighted sum of the objectives (accuracy, power consumption and MAC operations) with Zoph and Le (2017) models, for instance;

Dong et al. (2018) went further and used device awareness with Sequential model-based optimization (SMBO) in a cell structured search. It is worth mentioning that the search for their Pareto frontier took 8 GPU days on CIFAR10, even though they were using surrogate models;

Lu et al. (2020) used Genetic Algorithm to minimize two objectives: classification error and floating-point operations (FLOPs). The search took 4 *GPU Days* on CIFAR10 dataset;

Only a few differentiable objectives were explored so far: Cai, Zhu, and Han (2019) was able to differentiate latency using expected latency from the combination of the candidate models (LI; SUN, et al., 2021), i.e. a network to predict latency given the operators. To the best of our knowledge, there is no previous initiatives involving MOO on One-Shot NAS (mainly DARTS) with such clear differentiable formulation even in a multi-objective context, thus opening the possibility of speeding up the search.

3.2 Non-Inferior Set Estimation (NISE)

This technique (COHON; CHURCH; SHEER, 1979) is an iterative approach that uses the weighted sum method to automatically create, at the same time, an inner and an outer approximation of the Pareto frontier using a linear approximation. At every iteration, based on the already calculated efficient solutions, a segment between each neighboring pair of efficient solutions is traced, determining new weighting vectors. Then, the next $\mathbf{w}$ vector is selected using the highest error margin $\mu^{i,j}$ among all pairs on the Pareto frontier as shown on Figure 3.2.

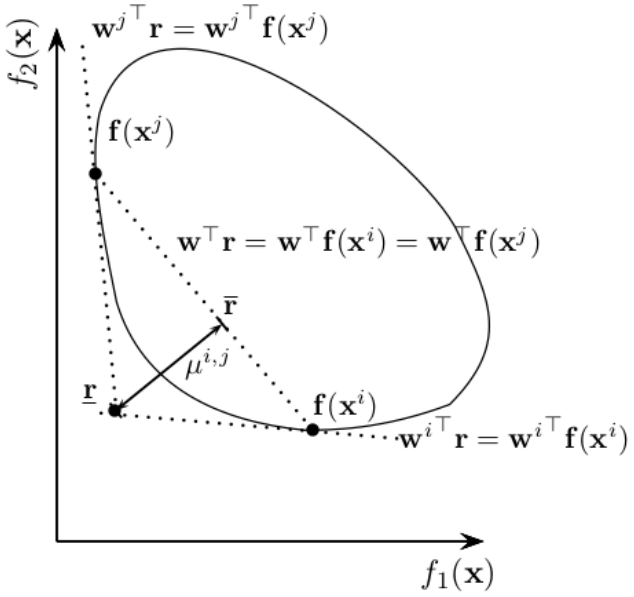


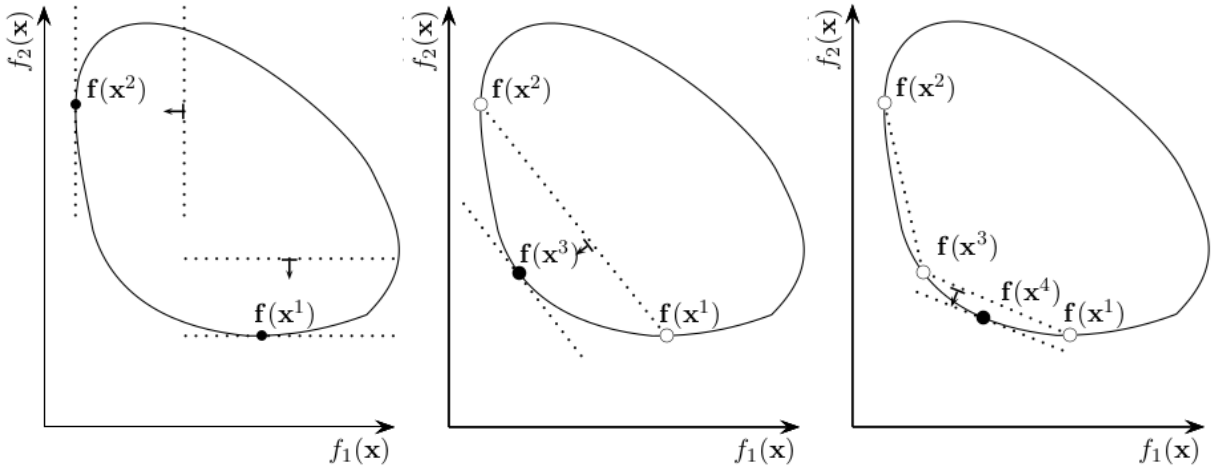
Figure 3.2 – NISE's calculation of the weight vector $\mathbf{w}$ and the error margin $\mu^{i,j}$. The weight vector $\mathbf{w}$ corresponds to the line that connects $\mathbf{f}(\mathbf{x}^i)$ and $\mathbf{f}(\mathbf{x}^j)$. The error margin $\mu^{i,j}$ is the distance between the minimal potentially achievable value $\underline{\mathbf{r}}$ and the current representation $\bar{\mathbf{r}}$. Extracted with permission from Raimundo (2018) Figure 4.

This procedure finds an accurate and fast approximation for problems with two objectives (ROMERO; REHMAN, 2003). Two neighboring efficient solutions (called neighborhood) are used to determine a new efficient solution employing the weighted sum method. The stopping criterion is defined to ensure the quality threshold of the approximation (RAIMUNDO; FERREIRA; ZUBEN, 2020). Figure 3.3 illustrates an iteration of this process. NISE was further extended by Raimundo, Ferreira, and Zuben (2020) for more than two objectives, given rise to MONISE (many-objectives NISE).

On the other hand, in the case of non-convex Pareto frontiers, efficient solutions dominated by convex combinations of other efficient solutions are not achievable by NISE, as it happens with all strategies based on the weighted sum method. This limitation has forced us to adapt NISE to find an approximation of the frontier in order to find Pareto-optimal solutions (see Subsection 3.3.1).

3.2.1 Weighted sum method

This method consists of a simple manner to implement a weighted combination of the objectives. Each component of the vector $\mathbf{w}$ represents the relative importance of each objective, or, in other words, it represents the preference for each objective. In our research, these weights are automatically determined and updated by NISE, with the purpose of populating the desired regions of the frontier (RAIMUNDO; FERREIRA;



(a) Initialization (b) First iteration (c) Second iteration

Figure 3.3 – Steps of the NISE algorithm. In (a) Initialization, $\mathbf{w}^1$ (horizontal line) and $\mathbf{w}^2$ (vertical line) are used to find the extreme solutions $\mathbf{f}(\mathbf{x}^1)$ and $\mathbf{f}(\mathbf{x}^2)$. In (b) First iteration, the weight vector $\mathbf{w}^3$ (established by the line that connects $\mathbf{f}(\mathbf{x}^1)$ and $\mathbf{f}(\mathbf{x}^2)$) is used to find $\mathbf{f}(\mathbf{x}^3)$. In (c) Second iteration, the weight vector $\mathbf{w}^4$ (established by the line that connects $\mathbf{f}(\mathbf{x}^1)$ and $\mathbf{f}(\mathbf{x}^3)$) is used to find $\mathbf{f}(\mathbf{x}^4)$, which is the choice that maximizes the margin.
Extracted with permission from Raimundo (2018) Figure 8.

ZUBEN, 2020). The method is defined as:

$$\begin{aligned} \min_{x} \quad & \mathbf{w}^\top \mathbf{f}(\mathbf{x}) \\ s.t. \quad & \mathbf{x} \in \Omega, \Omega \subset \mathbb{R}^n \\ & \mathbf{f}(\cdot) : \Omega \to \Psi, \Psi \subset \mathbb{R}^m \end{aligned} \tag{3.2}$$

where $\mathbf{w} \in \mathbb{R}^m$, $\sum_{i=1}^{m} \mathrm{w}_i = 1$ and $\mathrm{w}_i \geqslant 0 \ \forall i \in \{1, 2...m\}$.

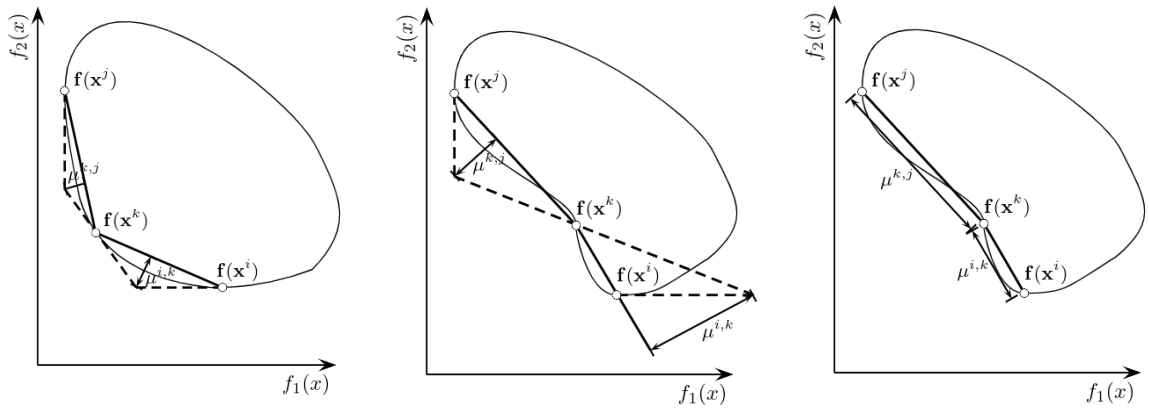
3.3 Sampling scalarized NAS solutions

NISE was the baseline adopted in our formulation due to its simplicity, deterministic nature and automatic distribution of the obtained efficient solutions along the frontier, using scalarization of loss and regularization terms, briefly illustrated in Figure 3.3.

3.3.1 Extending NISE to non-convex scenarios

NISE relies on the convexity of the frontier to estimate the error μ between pairs of current efficient solutions and then estimating the next weight $\mathbf{w}$. However, in a non-convex scenario, it is not possible to rely on it. NISE algorithm would fail for NAS

solutions because it might face non-convex frontiers while generating efficient solutions[1]. For instance $\mathbf{w}^{k^\top}$ is not a convex combination of $\mathbf{w}^{i^\top}$ and $\mathbf{w}^{j^\top}$ on Figure 3.4b, taken as a core step of NISE's operation (RAIMUNDO; FERREIRA; ZUBEN, 2020). In fact, in the case of non-convex Pareto frontiers, it is possible to obtain errors μ that are not related to the frontier (e.g., $\mu^{i,k}$ on Figure 3.4b). In other words, it can yield big errors for a small distance.



(a) Convex frontier (b) Non-convex frontier (c) Suitable behavior for a non-convex frontier

Figure 3.4 – Extending NISE algorithm for non-convex Pareto frontiers: $\mathbf{f}(\mathbf{x}^k)$ is an efficient solution but the Pareto frontier is not convex in Figure (b) and (c). In (b) $\mu^{i,k}$ is bigger than $\mu^{k,j}$ however the neighbourhood between $\mathbf{f}(\mathbf{x}^j)$ and $\mathbf{f}(\mathbf{x}^k)$ is less explored. This discrepancy is reduced in (c) using Euclidean distance metric to calculate μ.

Aiming at generalizing the formulation to non-convex scenarios, we calculate the error using a simple metric: the Euclidean distance between the solutions in a neighborhood. Besides that adaptation, the algorithm follows the same logic of NISE to find the weights using the following linear system:

$$\begin{cases} \mathbf{w}^\top \mathbf{f}(\mathbf{x}^i) = \mathbf{w}^\top \mathbf{f}(\mathbf{x}^j), \\ \sum_{i=1}^{2} \mathrm{w}_i = 1, \end{cases} \tag{3.3}$$

The steps of the algorithm to sample the weights of the Pareto frontier can be summarized with the following pseudo code:

[1]Be aware that those solutions could be non-optimal on a convex frontier, but without guarantees of optimality, the unknown shape of the frontier and the usage of proxies, we worked with the worst case scenario of a non-convex frontier.

Algorithm 3.1 NISE extension for non-convex scenarios

```
threshold ← 0.15
frontier ← [findSolution(1.0, 0.0), findSolution(0.0, 1.0)]        ▷ Equation (3.4)
distance ← ∞
while distance > threshold do
    max_d ← 0
    for i = 0 to length(frontier) − 1 do
        d ← euclideanDistance(frontier[i], frontier[i + 1])
        if d > max_d then
            max_d ← d
            w ← calculateWeight(frontier[i], frontier[i + 1])      ▷ Equation (3.3)
        end if
    end for
    distance ← max_d
    candidate ← findSolution(w)                                    ▷ Equation (3.4)
    if isParetoOptimal(candidate, frontier) then
        frontier ← frontier + candidate
        sort(frontier)          ▷ The frontier needs to be in order for d calculation
    end if
end while
```

The threshold represents the stopping criterion for the algorithm. It is calculated using the normalized values of the objectives to avoid one objective being considered more important than another. The *findSolution* method returns the 2D solution found using the weight **w** (parameter of the scalarization).

The non-optimality of solvers for non-convex problems enforces an adaptation that consists of discarding dominated solutions and reordering the solutions to calculate the weights. Given that, *isParetoOptimal* method checks if a solution belongs to the current approximation of the Pareto frontier, which means it cannot be bigger (on both dimensions at the same time) than any other point on the frontier. If the solution doesn't belong to the frontier, then it is discarded.

We created a constraint to guarantee that each **w** from one pair of parents (points from the frontier) is explored only once to avoid infinite loops. Thus, if that **w** yield a dominated point, the neighborhood is invalidated. Besides that, it is relevant to highlight that one candidate can dominate multiple points on the frontier at once, shrinking the frontier elements eventually.

3.3.2 Proposed Multi-objective NAS formulation

The weighted sum method and the gradient-based NAS formulation were combined in our proposal by means of a convex combination. Starting from the BLO formulation from DARTS to differentiate NAS (Eq. (2.5)), we arrive at a convex combination

of two conflicting objectives:

$$\boldsymbol{w}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{w}}\{(1-\nu)\cdot\mathcal{L}_{tr}(\boldsymbol{w},\boldsymbol{\alpha}) + \nu\cdot\mathcal{L}_{reg}(\boldsymbol{w})\} \tag{3.4}$$

where $\boldsymbol{w}$ are the weights of the neural network and $\mathbf{w} = [1-\nu, \nu]$ to comply with NISE (see Equation (3.3)).

The multi-objective procedure generates different weights ν ($0 \leq \nu \leq 1$), thus promoting the creation of multiple models, each one with a unique trade-off between accuracy ($\mathcal{L}_{tr}$) vs. complexity ($\mathcal{L}_{reg}$). Unfortunately applying regularization directly to architecture parameters ($\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$) is not possible because they are already a weighted sum with a fixed value.

In the next chapter, we will explain how we applied Eq. (3.4) on the Search stage of PC-DARTS in order to populate the Pareto frontier and find relevant results using the efficient models on it.

4 Methodology and Proposed Experiments

This research resulted in two theoretical developments to achieve our goals: (i) Formulation of NAS exploring MOO (see Subsec. 3.3.2): it consists of considering the loss function as conflicting with a regularization factor (L_1 or L_2); (ii) NISE adaptation for non-convex functions to generate the Pareto frontier (see Subsec. 3.3.1) using scalarization of the objectives in order to have distributed samples at a reasonable computational cost.

Using a NISE adaptation on a multi-objective NAS model generates a set of codes (base architecture cell structure encoded in a string) with loss vs. regularization trade-off. Given that, we designed a series of experiments to validate the following assumptions: (Experiment 1) is Alg. 3.1 capable of finding a good representation of Pareto frontier of NAS models? (Experiment 2) Is it possible to select the diverse models found by Alg. 3.1 without needing to execute PC-DARTS' evaluation stage for all codes? (Experiment 3) Is the model diversity of the (loss vs. regularization trade-off) Pareto frontier also diverse in accuracy, model size and latency? (Experiment 4) Can the aggregation of Pareto-optimal PC-DARTS' models improve accuracy performance?

4.1 Experimental analysis

The following experiments are according to PC-DARTS Search and Evaluation stages while iterating over each of them finding trade-offs of model complexity and accuracy (check Figure 4.1 for visual representation):

Experiment 1: Validation of multi-objective Search stage. In this experiment, we used Alg. 3.1 to estimate weights ν to populate the entire non-convex loss vs. regularization frontier, in which every solution of the frontier is obtained by the Search stage of PC-DARTS. Our goal in this experiment is to evaluate the capability of NISE's adaptation to create a representation of the Pareto-frontier for NAS models. Based on Eq. (3.4), we changed the original Search Stage loss calculation (sum of criterion loss) to a weighted sum of criterion and regularization loss weighted by 1-ν and ν respectively. We could select three different regularization cases (described on Section 4.2) in order to elect the most suitable regularization *a posteriori*. For each run of Search stage we found a proxy model and a code.

Experiment 2: Effect of multi-objective proxy models on Evaluation stage. In order to know the final performance of a code, we need to use it on the Evaluation stage to generate and train the final model with stacked cells. When training all efficient

models from the Pareto frontier using the codes from previous experiment (for each case) we obtained the real performance of each member of the frontier. Given that, the goal of this experiment is to find evidence of proxy features (accuracy, scalarization profile, or complexity of the model) that indicates the comparability with larger scale model performance (Evaluation stage result) (LIU, Y. et al., 2021). If we find such association we could avoid the cumbersome work to run each model in the Evaluation stage, that would be only necessary for the selected ones. We also compared the results between cases to elect the best one for this application accounting for the duration of **Experiment 1** and proxy quality. Given the insights obtained we developed a guideline to select a small subset of codes (of the case with best results) in order to find the most accurate models. This guideline is used to guide the following experiments.

Experiment 3: Evaluation of single models. Using the framework created (See Sections 5.4 and 5.5) we trained the most accurate models in order to check the best single model result and, at the same time, training a diverse pool of learning machines for the next experiment to reach the best possible outcome.

Experiment 4: Evaluation of ensemble model. An Ensemble was created using the models trained on **Experiment 3** given that proper representation of the Pareto frontier is a good sampling strategy for ensemble model generation, since it creates a diverse set of efficient models (in contrast to any other diverse set of models) (RAIMUNDO; DRUMOND, et al., 2021). The main purpose here is to demonstrate that the increase in complexity (with the ensemble of multiple efficient models) is compensated favorably by the improvement in performance.

4.2 Cases: Conflicting Objectives

The conflicting objectives used for sampling the frontier, restricting the model weights, are:

- Cross-entropy: default model loss function used to tune free parameters (weights) by means of the backpropagation process.
- L_2 or mean squared error: This loss function is used as $\mathcal{L}_{reg}(\boldsymbol{w}) = \sum_{i=1}^{n}(w_i)^2$ where n is the number of elements of the weight vector or matrix.
- L_1 or mean absolute error: This loss function is used as $\mathcal{L}_{reg}(\boldsymbol{w}) = \sum_{i=1}^{n}|w_i|$ where n is the number of elements of the weight vector or matrix.

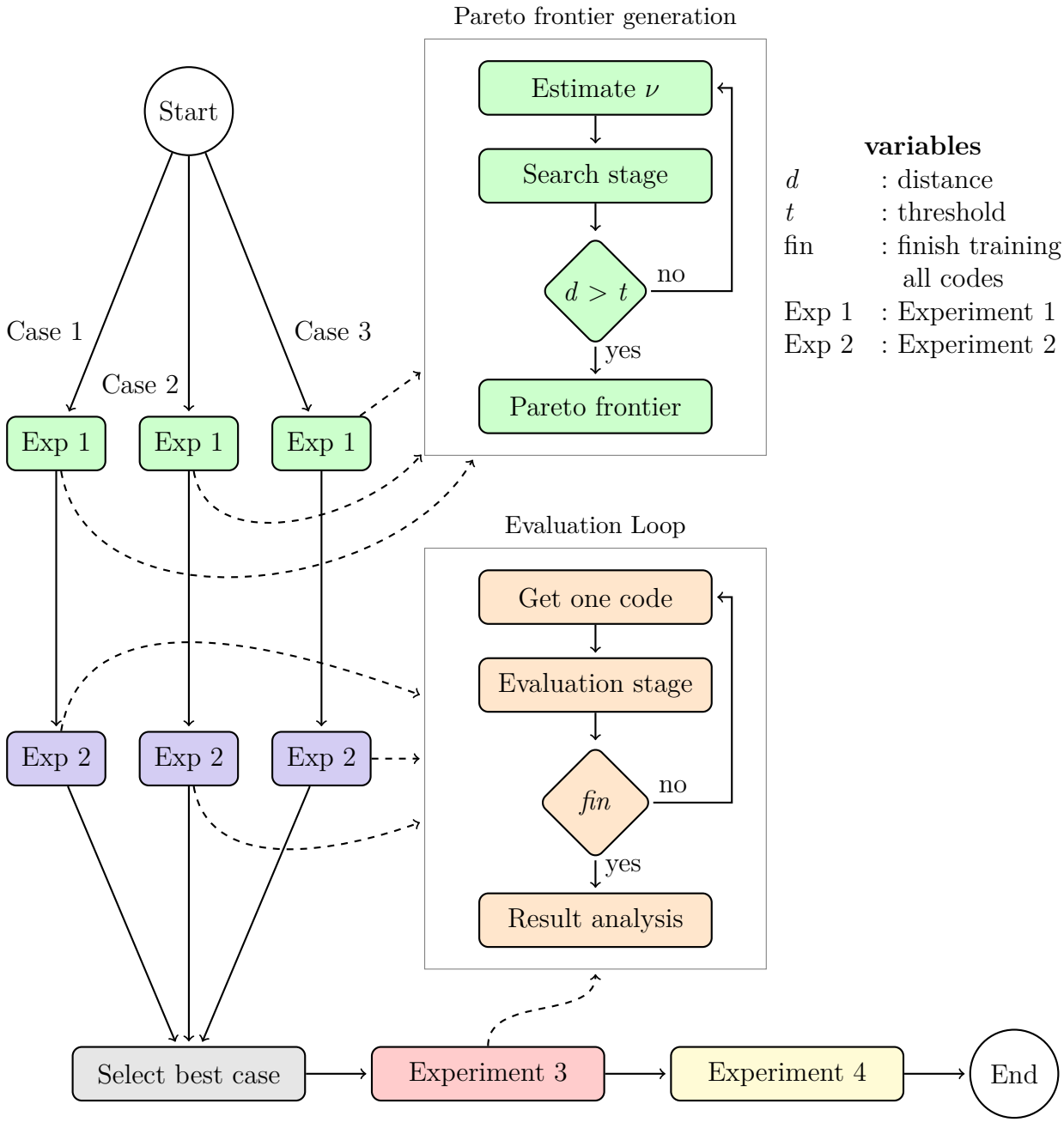


Figure 4.1 – Proposed flow of experimental scenarios. The output from **Experiment 1** is the input for **Experiment 2** case-wise. The **Experiment 3** is going to evaluate again the best codes of the best case. For more details about d and t please refer to Alg. 3.1

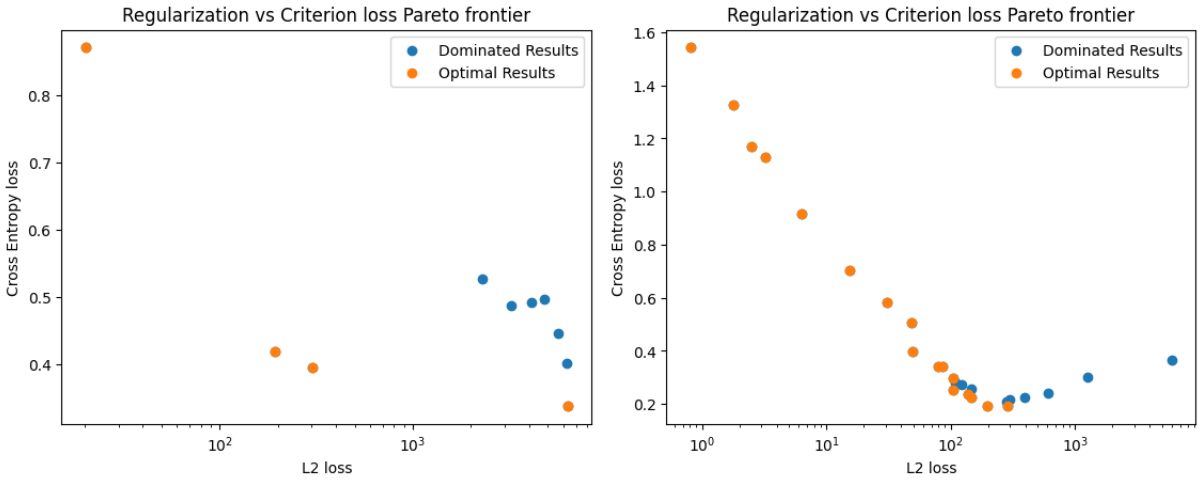
We decided to use L_2 and L_1 regularization because they are wide-spread functions to avoid overfitting and L_1 also promote feature pruning. We set an experiment using Cross-entropy vs. L_2 loss and another using Cross-entropy vs. L_1 loss.

Since L_2 is already present on the original baseline to help with optimization, acting like a penalty to generate smaller layer weights, this inspired us to create a third experiment using Cross-entropy versus L_1, while L_2 is fixed using baseline weight value (3e-4). Therefore, we have the following final three cases for **Experiments 1** and **2**:

1. Cross-entropy loss and L_2 loss objectives;
2. Cross-entropy loss and L_1 loss objectives;
3. Cross-entropy loss and L_1 loss objectives with fixed weight of L_2 loss;

5 Experimental Results

In this research, we present and analyze the results of the four experiments described in Section 4 using the baseline implementation PC-DARTS (XU et al., 2020) with minor code changes. We employed an automatic adaptive variation of the weights using Algorithm 3.1 to investigate the multi-objective nature of the model and understand the impact of distinct trade-off solutions between training loss vs. complexity of the model (L_1 and L_2 regularization). Figure 5.1 shows the improvement against preliminary tests using manual weight estimation, a commonly used method with fixed step (linear or exponential) rate to estimate hyperparameter values during grid-search.



(a) Manual (b) Automatic using Algorithm 3.1

Figure 5.1 – Comparison of manual and automatic scalarization methods.

Notice that there is no guarantee of achieving any point belonging to the Pareto frontier, given the non-convexity of the learning problem. Nonetheless, the search were able to provide candidate solutions to populate a multi-objective like Pareto frontier with valuable trade-offs of size and accuracy. It was also possible to obtain smaller models with competitive performance when compared to the baseline. The source code is available at `https://github.com/iksmada/MOOD-NAS`.

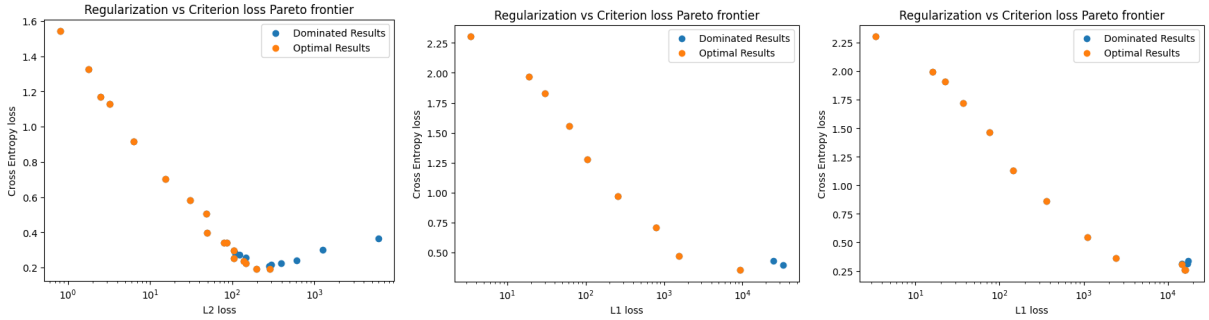
5.1 Initialization

After preliminary tests and given the nature of neural networks, it does not make sense to initialize the weights as proposed by NISE ($\nu = 0.0$ in Equation (3.4) that result in a model that only optimizes cross-entropy and $\nu = 1.0$ that only optimizes

regularization). Instead, we empirically selected $\nu = 0$ and $\nu = 0.2$ initial regularization loss weights for the cases because starting with $\nu = 1.0$ created a long loop until it starts generating Pareto-optimal solutions, inflating the computational burden.

Finally $\nu = 0.2$ proved to be a good approximation for the initialization since it generated, for our 3 cases, codes (base architecture cell structure encoded in a string) with weight-free operations (pooling or skip-connections). The model size (Params) of $\nu = 0.2$ on Tables 5.2, 5.3 and 5.4 attests that behavior. Another empirical tuned value was the threshold. We decided that *threshold*=0.15 on Alg. 3.1 was a reasonable value because it was able to populate the entire space using around 10 to 20 solutions on the frontier.

When training time is a constraint, it can be reduced (with sampling damage) using a higher threshold. It is worth mentioning that GPUs with smaller memory size run slower while new GPU architectures run faster.



(a) L2 loss (b) L1 loss (c) L1 loss with fixed L2 weight

Figure 5.2 – Pareto frontier population on the Search stage for all 3 cases.

5.2 Experiment 1: Validation of multi-objective Search stage

We ran one case at a time with two conflicting objectives looping over the Search stage (described on Subsec. 2.5.2): L_2 loss and cross-entropy loss; L_1 loss and cross-entropy loss; and L_1 loss and cross-entropy loss with a fixed value of L_2 loss. The duration of each case changed given the number of drawn samples, with Table 5.1 summarizing it. The configuration was the same from baseline, except small code changes to monitor loss:

- 50% of the data is used for model weights training and 50% for architecture weights training.
- The total number of epochs is limited to 50, out of which the first 15 were not used to train the architecture to alleviate the preference for parameterized operations.

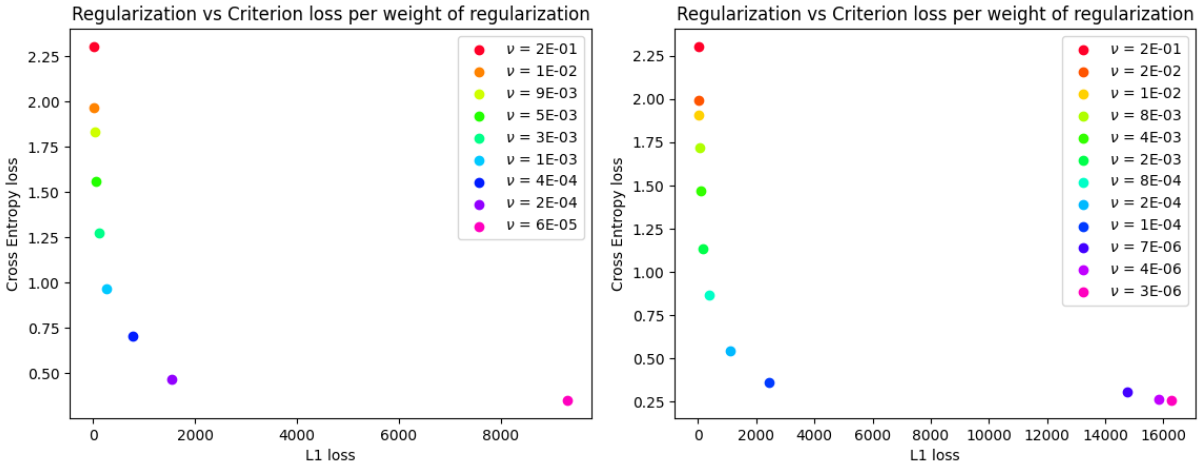
- The search step consists of stacking 8 cells (6 normal and 2 reductions cells on 1/3 and 2/3 of the whole network).
- Each cell has 6 nodes. The first and second nodes of cell k are set equal to the outputs of cell $k-2$ and cell $k-1$. See Figure 5.4.
- For partial connections we used $K = 4$, thus only 1/4 features are sampled on each edge, which yields almost 4 times bigger batch size.
- First-order approximation was used for $\boldsymbol{w}^*(\boldsymbol{\alpha})$.
- $\mathcal{O} = \{$ 3×3 and 5×5 separable *conv*, 3×3 and 5×5 dilated separable *conv*, 3×3 max pooling, 3×3 average pooling, identity (skip-connection), and zero (no connection) $\}$. This set is the same from the baselines.
- Tests used Tesla V100 GPU with 12 GB of memory.
- All tests ran on the CIFAR10 dataset.

Table 5.1 – **Experiment 1** duration per case. The Pareto-optimal ratio was calculated using the number of optimal solutions on the frontier divided by the *Model quantity* column.

Cases	Model quantity	Pareto-optimal ratio (%)	Time (GPU days)	
			Total	Average
L_2	26	65	4.4	0.17
L_1	11	82	1.9	0.17
L_1 with fixed L_2	15	80	2.2	0.15
PC-DARTS	1	N/A	N/A	0.16*

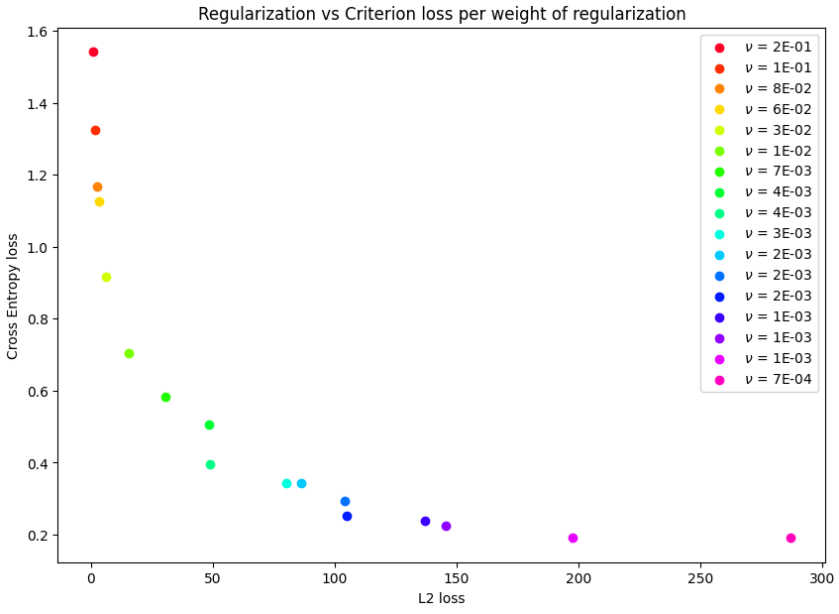
*For 10 runs

The experiment applied weighted sum regularization penalizing original cross-entropy loss using the proposed multi-objective NAS formulation on Eq. (3.4). By changing the weight ν we were able to sample multiple architecture codes (candidates in the search space). The response to the regularization objectives resulted in relevant trade-offs that support the motivations of this research and can be seen on Figure 5.3. The results will be discussed in the next section.



(a)

(b)



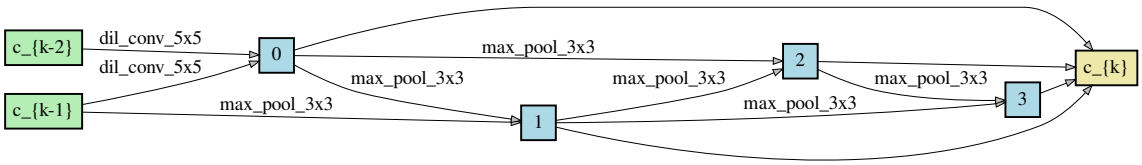
(c)

Figure 5.3 – PC-DARTS behavior along the approximated Pareto frontier applying regularization. (a) and (c) follow Eq. (3.4) using L_1 and L_2 loss as $\mathcal{L}_{reg}$ respectively. (b) Since it has a fixed L_2 value, Eq. (3.4) loss became: $(1-\nu)\cdot\mathcal{L}_{tr} + \nu\cdot L_1 + 3\text{e-}4\cdot L_2$.

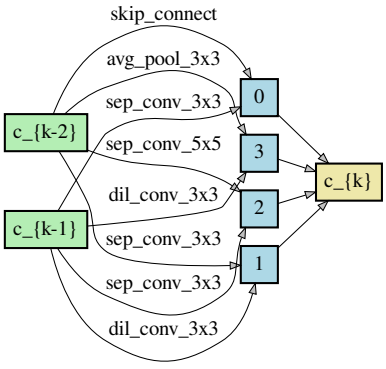
5.3 Experiment 2: Effect of multi-objective proxy models on Evaluation stage

The appearance of the frontier during Search stage was above expectation, but on the other hand the only output from it are the codes. For that reason we spent time training all samples from the frontiers to validate our findings and the proxies (i.e. Search stage results) reliability.

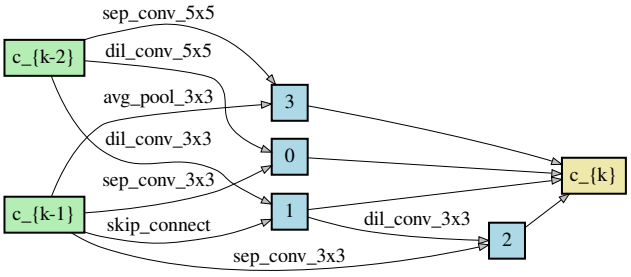
To train each code of the candidate solutions at the frontier we used the Evaluation stage (complete network defined by the edges with the highest weights only) configuration from the baseline without any relevant intervention, since each sample from the frontier has an architecture code of the normal and reduction cell, that can be easily visualized on Fig. 5.4. The network has 20 stacked cells (18 normal and 2 reduction cells as well) and was trained for 600 epochs. The training was made using cutout (DEVRIES; TAYLOR, 2017) and an auxiliary loss tower (XU et al., 2020) following the same settings from the baseline in order to have comparable results. At the same time we introduced a validation set with 10% of the training set to check its reliability for filtering methods when creating a diverse ensemble or imposing accuracy constraints.



(a) Fig. 5.3c ν = 3e-2



(b) PC-DARTS



(c) Fig. 5.3c ν = 1e-3 pink circle

Figure 5.4 – Comparison of the code of the best trade-offs and the baseline. It is worth mentioning that the smaller models, like (a) (See Table 5.2 $\nu = \{0.03, 0.0013\}$), has more weightless operations like `skip_connect`, `max_pool` or no connection.

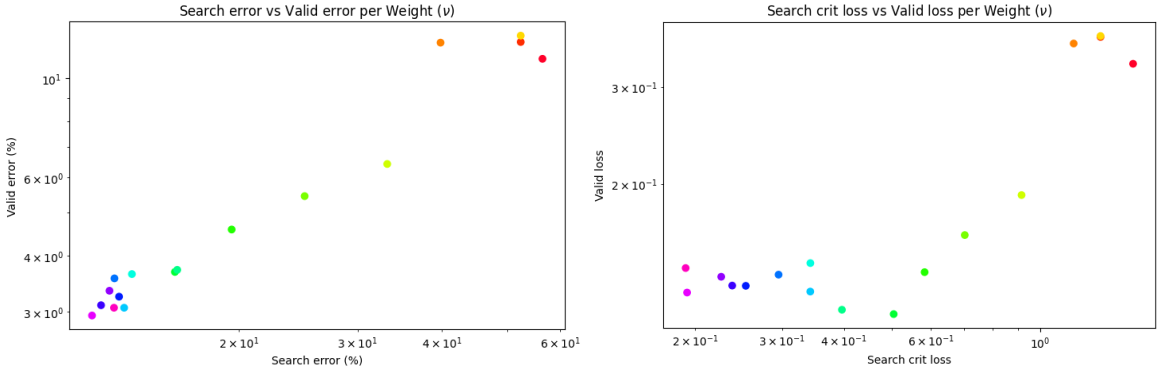
Each subsection will focus on each case results and analysis. At the end of those sections the framework and a guide will be introduced to help user decision.

5.3.1 L_2 loss

We evaluated all samples from the Pareto frontier of Figure 5.3c, the case using L_2 loss and cross-entropy loss. From the data generated we have drawn the following conclusions:

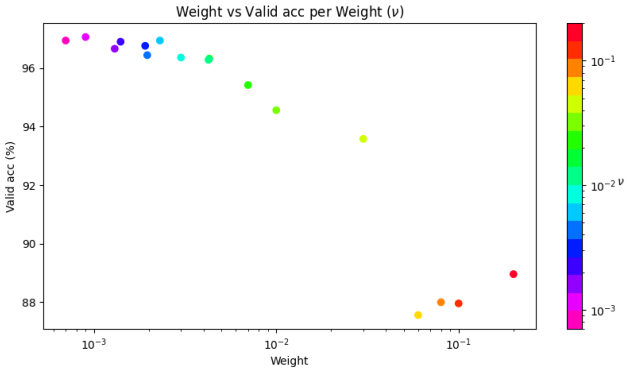
- The evaluation accuracy and search Accuracy (acc) had a high correlation (Check **Test acc** and **Search acc** on Figure 5.6), namely 97%, but the Figures 5.5a and b show a high uncertainty close to the smallest values of ν.
- Based on Figure 5.5c the weight had a direct effect on the accuracy so one can rely on it to select models from the proxies. Worth mentioning that such correlation does not indicate that lowest regularization would improve the performance, but that low regularization in the Pareto frontier improve performance.
- Some relevant trade-offs were found from the frontier when looking at Table 5.2:
  - for $\nu = 0.0013$ we have a competitive accuracy whereas it is 10% smaller than the baseline (check Table 5.5).
  - for $\nu = 0.03$ we have a 51% smaller model whereas it is only 4% less accurate than the baseline.

This test was also useful to check the reliability of a validation set on Evaluation stage encouraging its use as heuristic for ensemble generation or any other selection step. The original configuration uses all training data from CIFAR10 to train the model. The correlation between test and validation accuracy on Evaluation stage was 97%, showing that it is reliable if needed. This is also valid for the next subsections.



(a) Search stage accuracy

(b) Search stage Cross-entropy



(c) Weight

Figure 5.5 – Comparison of the behavior between search and validation set accuracy on Evaluation stage for L_2 regularization. Metrics on Search stage refers to the training set (half of the training samples used to train the model weights).

Table 5.2 – L_2 loss case Pareto frontier evaluation. Validation (Valid) and Test refers to the sets during Evaluation stage. All other data is known at the end of Search stage. Loss as the optimization criterion (*crit loss*) is cross-entropy loss and Loss as the regularization term (*reg loss*) is L_2 loss in this case. The latency for GPU is measured with batch size 8 on NVIDIA 1660Ti with Pytorch 1.8.1+cuDNN in miliseconds.

Weight	Params	FLOPs	Latency	Accuracy		
ν	(M)	($\times 10^8$)	GPU	Search	Valid	Test
0.2	1.37	2.3	8.51	43.55	88.96	88.02
0.1	1.37	2.3	8.72	47.60	87.96	87.56
0.08	1.37	2.3	11.74	60.17	88.00	87.91
0.06	1.37	2.3	11.92	47.60	87.56	86.98
0.03	1.78	2.9	12.71	66.81	93.58	93.13
0.01	2.25	3.7	17.76	74.98	94.56	93.99
0.007	3.19	5.1	26.77	80.50	95.42	95.17
0.00429	4.11	6.4	36.93	83.94	96.32	96.16
0.00425	3.99	6.3	36.46	83.82	96.28	96.07
0.003	4.66	7.4	42.05	86.14	96.36	96.34
0.0023	4.25	6.6	39.53	86.50	96.94	96.72
0.00196	4.07	6.3	36.20	86.94	96.44	96.23
0.00191	4.44	6.9	40.26	86.74	96.76	96.64
0.0014	3.80	5.9	34.27	87.53	96.90	96.94
0.0013	3.29	5.2	24.03	87.16	96.66	96.85
0.0009	3.81	5.8	34.41	87.91	97.06	97.08
0.0007	3.99	6.2	45.41	86.96	96.94	96.76

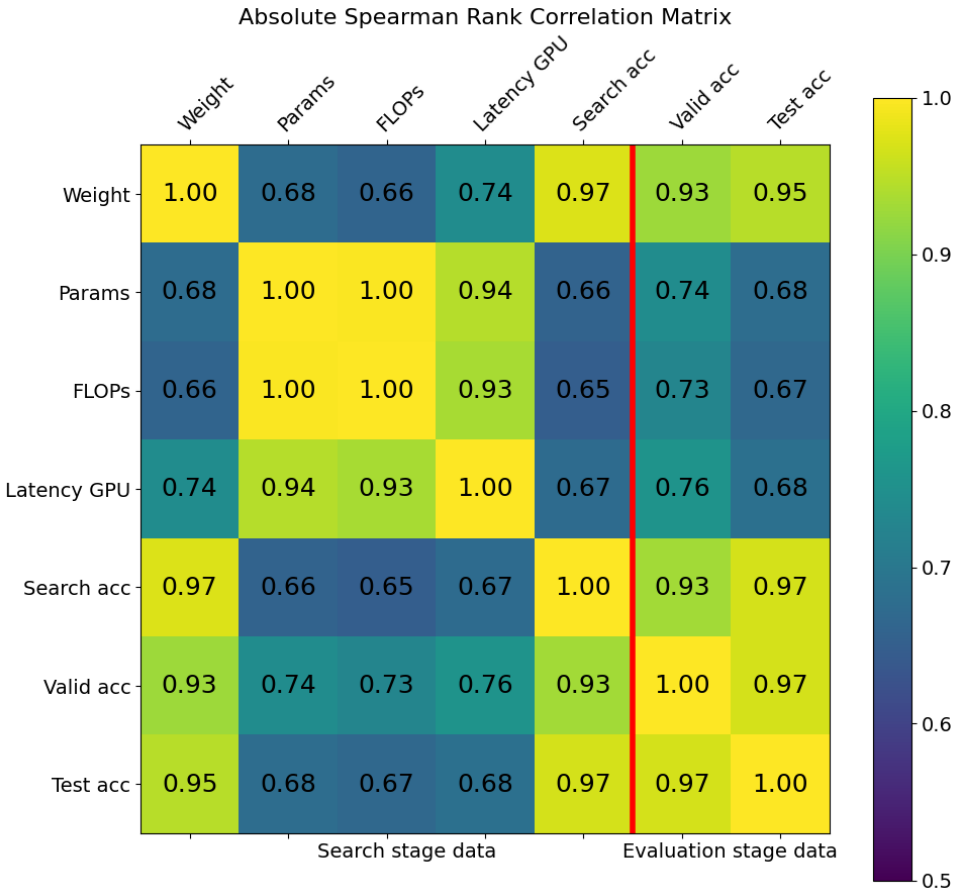


Figure 5.6 – Correlation matrix for L_2 regularization case. Table 5.2 was used to calculate it. The data gathered from Search stage is to the left of the red line. We selected Spearman Rank (KOKOSKA; ZWILLINGER, 2000) because it works better for logarithm relations since it ranks the values.

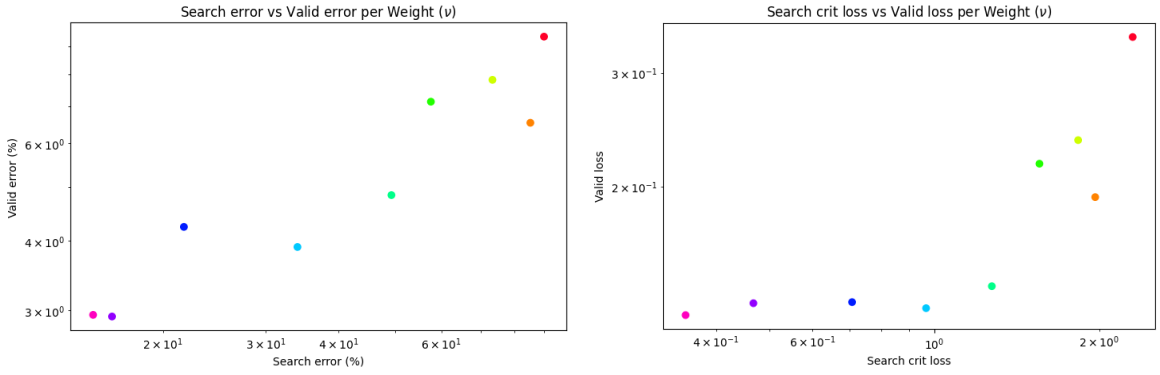
5.3.2 L_1 loss

We evaluated all samples from the Pareto frontier of Figure 5.3a, the case using L_1 loss and cross-entropy loss. From the data generated we have drawn the following conclusions:

- The evaluation accuracy and search accuracy hasn't as high correlation as the L_2 case (Check **Test acc** and **search acc** on Figure 5.8), namely 92%. Figures 5.7a and b show a correlation between axis, although the values are scattered. That way we cannot rely entirely on the search acc.
- Based on Figure 5.7c the weight had an effect on accuracy, but not as correlated as the case before.
- The correlation between test and validation accuracy on Evaluation stage was 100%, showing that it is reliable when used in a model selection step.
- Some relevant trade-offs were found from the frontier when looking at Table 5.3:
  - for $\nu = 0.00006$ we have a competitive accuracy whereas it is 15% smaller than the baseline (check Table 5.5).
  - for $\nu = 0.01$ we have a 52% smaller model whereas it is only 4.3% less accurate than the baseline.

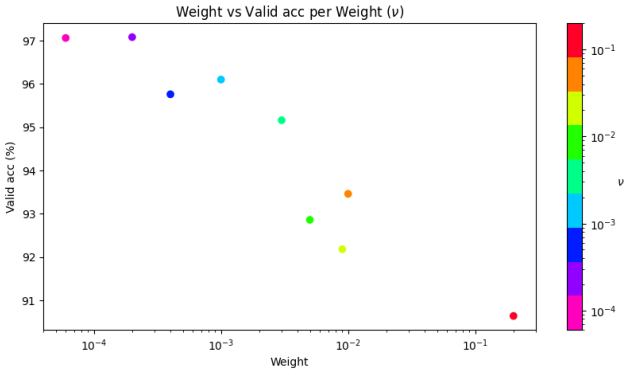
Table 5.3 – L_1 loss case Pareto frontier evaluation. Valid and Test refers to the sets during Evaluation stage. All other data is known at the end of Search stage. *crit loss* is cross-entropy loss and *reg loss* is L_1 loss in this case. The latency for GPU is measured with batch size 8 on NVIDIA 1660Ti with Pytorch 1.8.1+cuDNN in miliseconds.

Weight	Params	FLOPs	Latency	Accuracy		
ν	(M)	($\times 10^8$)	GPU	Search	Valid	Test
0.2	1.42	2.3	7.32	10.08	90.64	90.10
0.01	1.72	2.7	12.31	14.80	93.46	93.16
0.009	1.45	2.4	12.16	26.63	92.18	91.60
0.005	1.66	2.7	10.73	42.44	92.86	92.32
0.003	2.56	3.9	20.65	50.74	95.16	94.55
0.001	3.10	4.9	26.44	65.99	96.10	95.80
0.0004	4.03	6.5	33.03	78.27	95.76	95.55
0.0002	3.73	6.0	33.60	83.63	97.08	96.89
0.00006	3.08	4.8	24.45	84.81	97.06	96.79



(a) Search stage accuracy

(b) Search stage Cross-entropy



(c) Weight

Figure 5.7 – Comparison of the behavior between search and validation set accuracy on Evaluation stage for L_1 regularization with fixed L_2. Metrics on Search stage refers to the training set (half of the training samples used to train the model weights).

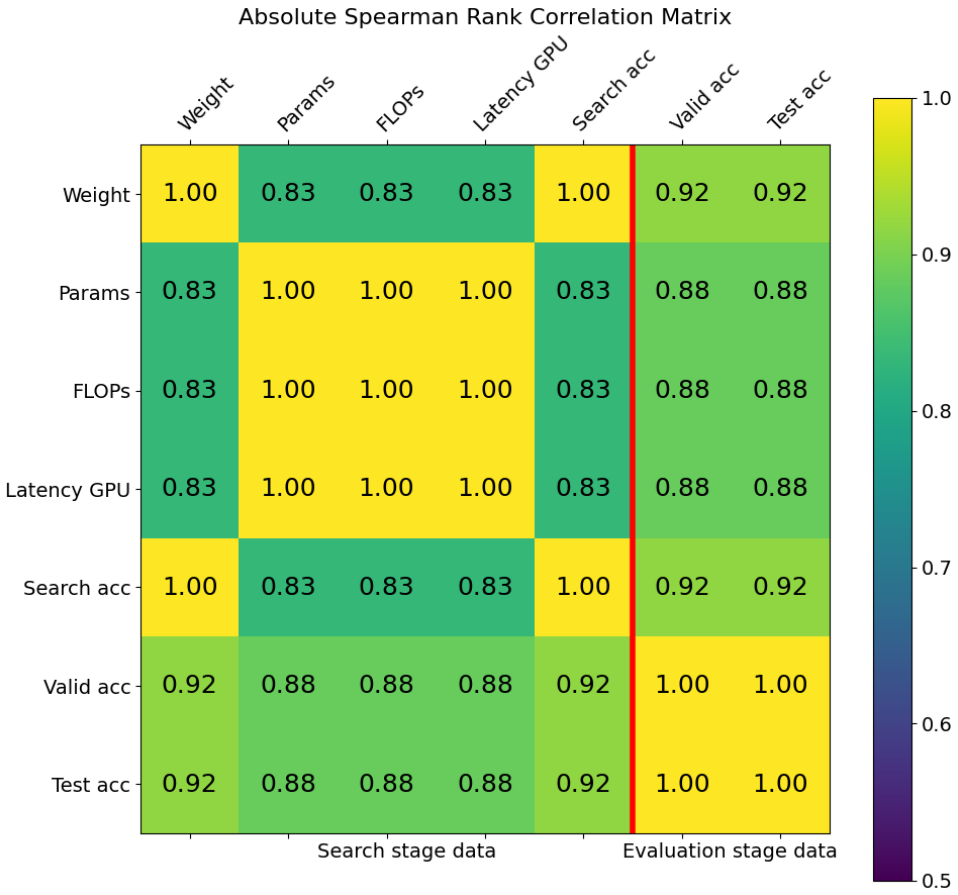


Figure 5.8 – Correlation matrix for L_1 regularization case. Table 5.3 was used to calculate it. The data gathered from Search stage is to the left of the red line. We selected Spearman Rank (KOKOSKA; ZWILLINGER, 2000) because it works better for logarithm relations since it ranks the values.

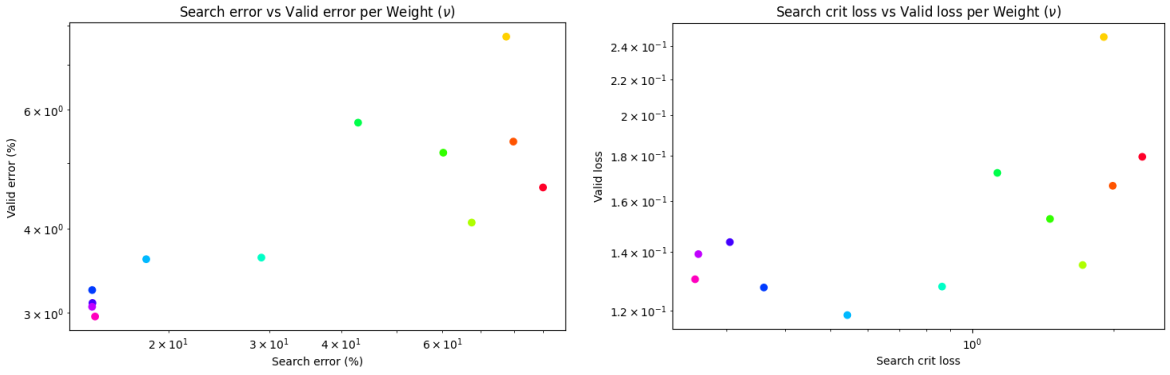
5.3.3 L_1 loss with fixed L_2

We evaluated all samples from the Pareto frontier of Figure 5.3b, the case using L_1 loss and cross-entropy loss with L_2 loss weight from the baseline. From the data generated we have drawn the following conclusions:

- The evaluation accuracy and search accuracy was the worst among cases(Check **Test acc** and **search acc** on Figure 5.10), namely 83%. Figures 5.9a and b supports it based on how the candidate solutions are spread out.
- Based on Figure 5.9c the weight had a direct effect on the accuracy for small ν values, so one can rely on it to select the most accurate models from the proxies.
- Some relevant trade-offs were found from the frontier when looking at Table 5.4:
  - for $\nu = 0.000003$ we have a competitive accuracy whereas it is 15% smaller than the baseline (check Table 5.5).
  - for $\nu = 0.2$ we have a 52% smaller model whereas it is only 2.7% less accurate than the baseline.

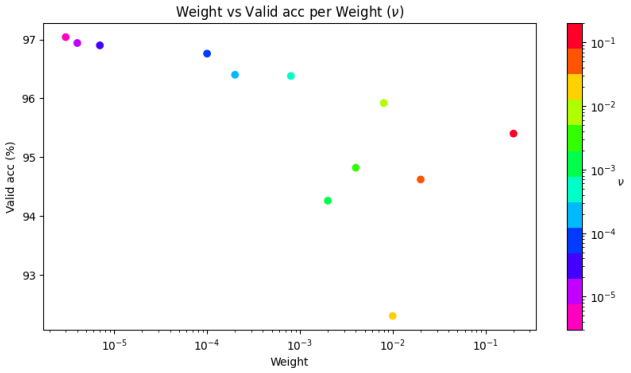
Table 5.4 – L_1 loss with fixed L_2 case Pareto frontier evaluation. Valid and Test refers to the sets during Evaluation stage. All other data is known at the end of Search stage. *crit loss* is cross-entropy loss and *reg loss* is L_1 loss in this case. The latency for GPU is measured with batch size 8 on NVIDIA 1660Ti with Pytorch 1.8.1+cuDNN in miliseconds.

Weight ν	Params (M)	FLOPs ($\times 10^8$)	Latency GPU	Accuracy		
				Search	Valid	Test
0.2	1.73	2.8	13.16	10.08	95.40	94.74
0.02	1.81	2.9	13.77	20.17	94.62	94.20
0.01	1.40	2.3	9.02	22.42	92.30	91.29
0.008	2.33	3.7	19.74	32.45	95.92	95.51
0.004	2.56	3.9	21.31	39.73	94.82	94.46
0.002	1.82	2.9	17.58	57.18	94.26	93.42
0.0008	3.41	5.6	29.70	70.94	96.38	95.66
0.0002	3.97	6.4	42.46	81.69	96.40	96.12
0.0001	4.25	6.6	39.42	85.26	96.76	96.55
0.000007	3.28	5.1	28.46	85.24	96.90	96.71
0.000004	3.99	6.3	36.68	85.26	96.94	96.90
0.000003	3.10	4.8	26.67	85.09	97.04	96.71



(a) Search stage accuracy

(b) Search stage Cross-entropy



(c) Weight

Figure 5.9 – Comparison of the behavior between search and validation set accuracy on Evaluation stage for L_1 regularization with fixed L_2. Metrics on Search stage refers to the training set (half of the training samples used to train the model weights).

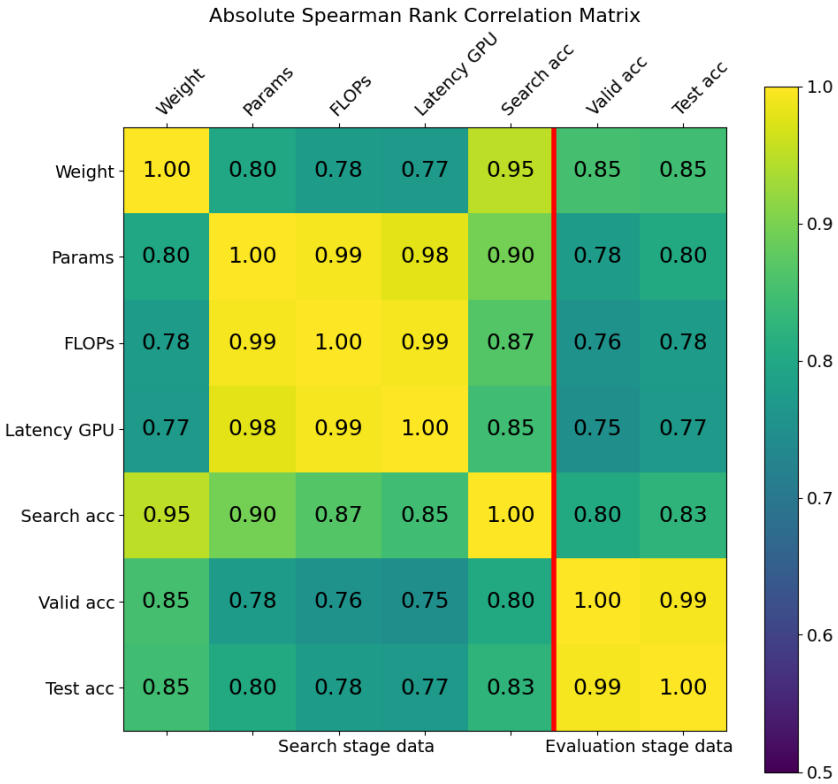


Figure 5.10 – Correlation matrix for L_1 regularization with fixed L_2 loss case. Table 5.4 was used to calculate it. The data gathered from Search stage is to the left of the red line. We selected Spearman Rank (KOKOSKA; ZWILLINGER, 2000) because it works better for logarithm relations since it ranks the values.

5.3.4 Cases evaluation

Experiment 2 results outlined in the previous subsections showed us that the frontiers using L_1 loss objective with fixed L_2 were faster than using L_2 loss given the bigger optimal solution ratio (Tab. 5.1). L_1 loss without L_2 loss were even faster but less stable (compare Fig. 5.7 and 5.9) than the case using fixed L_2 loss. For that reason the case of L_1 loss with fixed L_2 loss is more appropriate to use on the framework presented on Figure 5.11.

5.4 Framework Based on Results

We created a framework where the user can explore the diversity of the frontier according to the practical scenario while applying constraints after Search and Evaluation stage results. The proposed framework will generate the Pareto frontier for L_1 loss with fixed L_2 loss case only (like **Experiment 1**). Given the results, the user needs to train the suitable models according to the rules learned on **Experiment 2** (summarized on Section 5.5) and finally create an ensemble or select specific models. Figure 5.11 represents this framework.

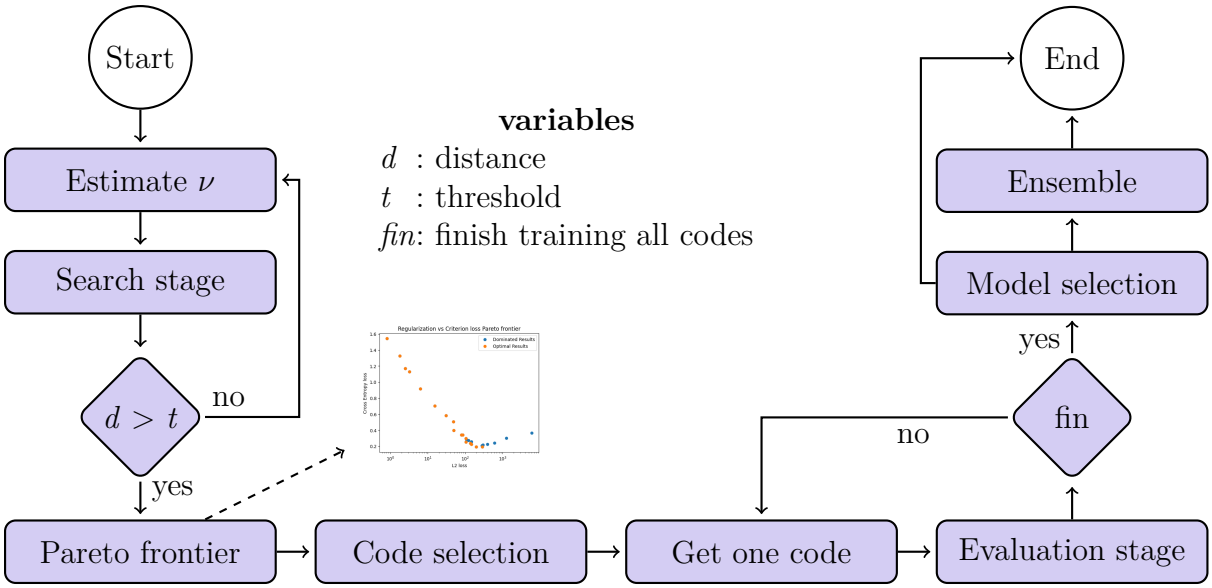


Figure 5.11 – Proposed framework for user application. For more details about d and t please refer to Alg. 3.1.

Most part of this framework is automatic using the source code available. On the other hand, two steps are manual and need attention given the user requirements:

- Code selection: A lot of metrics are available after the frontier is populated, namely: number of parameters, model size, FLOPs and latency. It is suitable to apply at that point any constraints in order to avoid infeasible training models.

- Model selection: After Evaluation stage the final performance is available making the selection more reliable since the proxies are approximations and comparable models to save time and resources.

5.5 User Guideline

When running the framework for specific intent some configuration can be tweaked. The main highlights are:

- Most accurate model: evaluate the codes of around 5 smallest weights ν of the frontier without validation set. Compare the accuracy among them after training to proceed to the final selection;
- Apply constraints: model related metrics (like Params, FLOPs or latency) can be used to filter codes before evaluating, thus saving time;
- Filtering: for model filtering focused on ensemble creation or other matters that require data analysis after training, one need to train all models, similar to Section 5.3, with validation set;
- Best trade-offs: the highlighted trade-offs were only spotted after evaluating all models, thus evaluating all codes. Validation set can be removed.

5.6 Experiment 3: Evaluation of single models

Since **Experiment 1** has the same first 4 steps from the framework shown on the previous sections, we decided to use the Pareto frontier from the case L_1 loss with fixed L_2 to do *a posteriori* code selection and make models and plots shared on this entire chapter.

We selected and evaluated the codes of the 5 smallest weights using no validation set in order to obtain the best accuracy possible and compare those with the baseline. Namely we used the weights $\nu = \{0.000003, 0.000004, 0.000007, 0.0001, 0.0002\}$ from Table 5.4 or visually the weights $\nu = \{3\text{e-}6, 4\text{e-}6, 7\text{e-}6, 1\text{e-}4, 2\text{e-}4\}$ from Figure 5.9 following the instructions from our guideline.

The best result locally achieved using PC-DARTS code (Fig. 5.4b) was 97.31%, a value that Xu et al. (2020) also claim to achieve on their paper on one of their 5 runs, but this is normal since we are using different GPUs (yielding different batch sizes and internal calculation) and the stochastic initialization by itself discussed on Section 3.1.

Although our results achieved a close performance with the one produced by the baseline, considering individual models, some of our results were able to reduce the gap between training and test accuracy and to generate smaller models yet competitive. Check Table 5.5 for more information.

Table 5.5 – Experiment 3 trained codes. The training was made using cutout and an auxiliary loss tower (XU et al., 2020). The ensemble is composed of all models from the list except PC-DARTS and $\nu = 2\text{e-}4$.

Characteristics	Train Acc	Test Acc	Params (M)
PC-DARTS *	98.98	97.31	3.63
Fig. 5.3b $\nu = 3\text{e-}6$	98.82	96.85	3.10
Fig. 5.3b $\nu = 4\text{e-}6$	97.15	96.82	3.99
Fig. 5.3b $\nu = 7\text{e-}6$	96.79	96.86	3.28
Fig. 5.3b $\nu = 1\text{e-}4$	96.43	97.02	4.25
Fig. 5.3b $\nu = 2\text{e-}4$	91.95	96.18	3.97
Ensemble	99.54	97.42	14.62

* Our reproduction.

5.7 Experiment 4: Evaluation of ensemble models

An ensemble was created using the codes of the 5 smallest weights from L_1 loss with fixed L_2 to test the improvement on performance. We assigned weights to the models using the validation dataset with a voting system. Each model received a vote when it predicted correcly while some other member of the committee don't. Finally the votes where normalize to weights between 0 and 1. The model related to the weight $\nu = 2\text{e-}4$ received 0 votes, so it was removed from the final accuracy estimation. We used a simple weighted sum of the last layer classes probabilities of all models to calculate the predictions. Test accuracy improved when compared to the results of our reproduction of the baseline, at the cost of a model four times bigger.

Concluding remarks

The first contribution of this research is the explicit formulation of the learning problem as a Multi-objective optimization problem. Many works took into account the model size or system latency (CAI; ZHU; HAN, 2019; CAI; GAN, et al., 2020) but without exploring the existence of a set of efficient candidate solutions at, or near, the Pareto frontier. Our research then aimed to explore the improvements that can be achieved by sampling multiple efficient candidate solutions, which are promoted by a multi-objective approach, with applications to both single model selection as well as to ensemble generation.

The second contribution is the algorithm to explore efficiently the approximation of the Pareto frontier adapting during iterations in order to explore automatically regions of interest with efficient solutions even for the non-convex neighbourhood.

The third contribution is a framework to train networks using this novel MOO formulation over a differentiable NAS outlined on Figure 5.11 and available with implementation detail at `https://github.com/iksmada/MOOD-NAS`. This allows the user to select the best code that fits the intended application based on memory, latency or size constraints, thus tuning automatically hyperparameters.

The last brings to the user more freedom and choice when training a NAS solution not based solely on one expert hyperparameter tuning, but with a relevant number of model trade-offs using an adaptive algorithm sampler together with a fast search procedure, given the high cost of NAS.

Future Work

The next straightforward step is to apply this framework to other datasets ready on the baseline, like CIFAR100 and ImageNet. Moreover, PC-DARTS also claim that CIFAR10 codes are transferable to ImageNet Evaluation stage.

We also intend to apply this same framework on different baselines. Since this multi-objective approach requires only a small amount of changes on the Search stage, we can apply our multi-objective proposal to almost any other NAS work with proxies.

At the same time we also consider to use 3 or more objectives with NISE extensions (RAIMUNDO; FERREIRA; ZUBEN, 2020) since it is clear from our results the conflicting objectives present on NAS formulation on Eq. (3.4).

Finally we want to combine different cells on the same network sequentially, given the cell diversity found on the frontier, creating a novel solution on the NAS field. This can overcome the barrier between cell and global space search to some intermediate path that can improve the lack of freedom in cell space search and a plethora of combinations provided by global space search.

References

BAKER, Bowen; GUPTA, Otkrist; NAIK, Nikhil; RASKAR, Ramesh. Designing Neural Network Architectures using Reinforcement Learning. **arXiv preprint arXiv:1611.02167**, 2016. arXiv: 1611.02167. Cited 2 times on pages 24.

BENGIO, Yoshua; COURVILLE, Aaron; VINCENT, Pascal. Representation learning: a review and new perspectives. **IEEE transactions on pattern analysis and machine intelligence**, v. 35, n. 8, p. 1798–1828, Aug. 2013. ISSN 0162-8828. DOI: 10.1109/tpami.2013.50. Cited 1 time on page 21.

CAI, Han; GAN, Chuang; WANG, Tianzhe; ZHANG, Zhekai; HAN, Song. Once for All: Train One Network and Specialize it for Efficient Deployment. In: ICLR 2020. **International Conference on Learning Representations**. [S.l.: s.n.], 2020. Available from: <https://arxiv.org/pdf/1908.09791.pdf>. Cited 2 times on pages 24, 62.

CAI, Han; ZHU, Ligeng; HAN, Song. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In: ICLR 2019. **International Conference on Learning Representations**. [S.l.: s.n.], 2019. Available from: <https://arxiv.org/pdf/1812.00332.pdf>. Cited 3 times on pages 24, 34, 62.

CHEN, Xin; XIE, Lingxi; WU, Jun; TIAN, Qi. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In: ICCV 2019. **2019 IEEE/CVF International Conference on Computer Vision**. [S.l.: s.n.], 2019. P. 1294–1303. Cited 1 time on page 28.

COHON, Jared L; CHURCH, Richard L; SHEER, Daniel P. Generating multiobjective trade-offs: An algorithm for bicriterion problems. **Water Resources Research**, Wiley Online Library, v. 15, n. 5, p. 1001–1010, 1979. Cited 2 times on pages 18, 34.

DEVRIES, Terrance; TAYLOR, Graham W. Improved Regularization of Convolutional Neural Networks with Cutout. **arXiv preprint arXiv:1708.04552**, 2017. arXiv: 1708.04552. Cited 1 time on page 26, 48.

DONG, Jin-Dong; CHENG, An-Chieh; JUAN, Da-Cheng; WEI, Wei; SUN, Min. DPP-Net: Device-aware Progressive Search for Pareto-optimal Neural Architectures. In: ECCV 2018. **Proceedings of the European Conference on Computer Vision**. [S.l.: s.n.], Sept. 2018. Cited 1 time on page 34.

ELSKEN, Thomas; METZEN, Jan Hendrik; HUTTER, Frank. Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution. In: ICLR 2019. **International Conference on Learning Representations**. [S.l.: s.n.], 2019. Available from: <https://openreview.net/forum?id=ByME42AqK7>. Cited 1 time on page 34.

GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: AISTATS 2010. **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010. v. 9, p. 249–256. Available from: <https://proceedings.mlr.press/v9/glorot10a.html>. Cited 1 time on page 22.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org. Cited 4 times on pages 21, 22, 23.

GREEN, Sam; VINEYARD, Craig M.; HELINSKI, Ryan; KOC, Cetin Kaya. RAPDARTS: Resource-Aware Progressive Differentiable Architecture Search. In: IJCNN 2020. **International Joint Conference on Neural Networks**. [S.l.: s.n.], 2020. P. 1–7. DOI: 10.1109/IJCNN48605.2020.9206969. Cited 1 time on page 28.

HE, Chaoyang; YE, Haishan; SHEN, Li; ZHANG, Tong. MiLeNAS: Efficient neural architecture search via mixed-level reformulation. In: CVPR 2020. **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. P. 11993–12002. Cited 1 time on page 27.

HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep Residual Learning for Image Recognition. In: CVPR 2016. **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. P. 770–778. DOI: 10.1109/CVPR.2016.90. Cited 3 times on pages 22, 23.

HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert. Multilayer feedforward networks are universal approximators. **Neural Networks**, v. 2, n. 5, p. 359–366, 1989. ISSN 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. Available from: <https://www.sciencedirect.com/science/article/pii/0893608089900208>. Cited 1 time on page 19.

HSU, Chi-Hung; CHANG, Shu-Huan; LIANG, Jhao-Hong; CHOU, Hsin-Ping; LIU, Chun-Hao; CHANG, Shih-Chieh; PAN, Jia-Yu; CHEN, Yu-Ting; WEI, Wei; JUAN, Da-Cheng. MONAS: Multi-Objective Neural Architecture Search using Reinforcement Learning. **arXiv preprint arXiv:1806.10332**, 2018. arXiv: 1806.10332. Cited 1 time on page 34.

IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: ICML'15. **Proceedings of the 32nd International Conference on International Conference on Machine Learning**. Lille, France: JMLR.org, 2015. v. 37, p. 448–456. Cited 1 time on page 22.

JIANG, Yufan; HU, Chi; XIAO, Tong; ZHANG, Chunliang; ZHU, Jingbo. Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition. In: EMNLP-IJCNLP. **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing**. Hong Kong, China: Association for Computational Linguistics, Nov. 2019. P. 3585–3590. DOI: 10.18653/v1/D19-1367. Available from: <https://www.aclweb.org/anthology/D19-1367>. Cited 1 time on page 28.

JIN, Yaochu; SENDHOFF, Bernhard. Pareto-based multiobjective machine learning: An overview and case studies. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 38, n. 3, p. 397–415, 2008. Cited 1 time on page 17.

KIM, Ye-Hoon; REDDY, Bhargava; YUN, Sojung; SEO, Chanwon. NEMO : Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy. **Journal of Machine Learning Research: Workshop and Conference Proceedings**, 2017. Cited 1 time on page 34.

KOKOSKA, Stephen; ZWILLINGER, Dan. **CRC Standard Probability and Statistics Tables and Formulae, Student Edition**. [S.l.: s.n.], Mar. 2000. ISBN 9781482273847. DOI: 10.1201/b16923. Cited 0 times on pages 52, 55, 58.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In: NIPS CONFERENCE. **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2012. v. 25. Available from: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>. Cited 1 time on page 20.

LI, Liam; TALWALKAR, Ameet. Random Search and Reproducibility for Neural Architecture Search. **arXiv preprint arXiv:1902.07638**, 2019. arXiv: 1902.07638. Cited 1 time on page 25.

LI, Siyi; SUN, Yanan; YEN, Gary G.; ZHANG, Mengjie. Automatic Design of Convolutional Neural Network Architectures Under Resource Constraints. **IEEE Transactions on Neural Networks and Learning Systems**, p. 1–15, 2021. DOI: 10.1109/TNNLS.2021.3123105. Cited 1 time on page 34.

LIU, Hanxiao; SIMONYAN, Karen; YANG, Yiming. DARTS: Differentiable ARchiTecture Search. In: ICLR 2019. **International Conference on Learning Representations**. [S.l.: s.n.], 2019. Available from: <https://openreview.net/forum?id=S1eYHoC5FX>. Cited 7 times on pages 25, 27, 28, 29.

LIU, Risheng; GAO, Jiaxin; ZHANG, Jin; MENG, Deyu; LIN, Zhouchen. Investigating Bi-Level Optimization for Learning and Vision from a Unified Perspective: A Survey and Beyond. **arXiv preprint arXiv:2101.11517**, 2021. arXiv: 2101.11517. Cited 1 time on page 27.

LIU, Yuqiao; SUN, Yanan; XUE, Bing; ZHANG, Mengjie; YEN, Gary G.; TAN, Kay Chen. A Survey on Evolutionary Neural Architecture Search. **IEEE Transactions on Neural Networks and Learning Systems**, p. 1–21, 2021. DOI: 10.1109/TNNLS.2021.3100554. Cited 1 time on page 26, 41.

LU, Zhichao; WHALEN, Ian; DHEBAR, Yashesh; DEB, Kalyanmoy; GOODMAN, Erik; BANZHAF, Wolfgang; BODDETI, Vishnu Naresh. NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm (Extended Abstract). In: IJCAI-20. **Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], July 2020. P. 4750–4754. Sister Conferences Best Papers. DOI: 10.24963/ijcai.2020/659. Cited 1 time on page 34.

MARQUES, Alan Caio Rodrigues. **Contribuição a Abordagem de Problemas de Classificação por Redes Convolucionais Profundas**. 2018. PhD thesis – University of Campinas - School of Electrical and Computer Engineering. DOI: 10.47749/T/UNICAMP.2018.991774. Cited 2 times on pages 21.

MCCULLOCH, Warren; PITTS, Walter. A Logical Calculus of Ideas Immanent in Nervous Activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 127–147, 1943. Cited 1 time on page 19.

MIETTINEN, Kaisa. **Nonlinear multiobjective optimization**. [S.l.]: Springer Science & Business Media, 2012. v. 12. Cited 1 time on page 32.

NAIR, Vinod; HINTON, Geoffrey E. Rectified Linear Units Improve Restricted Boltzmann Machines. In: ICML'10. **Proceedings of the 27th International Conference on International Conference on Machine Learning**. Haifa, Israel: Omnipress, 2010. P. 807–814. ISBN 9781605589077. Cited 1 time on page 22.

PASCANU, Razvan; MIKOLOV, Tomas; BENGIO, Yoshua. On the Difficulty of Training Recurrent Neural Networks. In: ICML'13. **Proceedings of the 30th International Conference on International Conference on Machine Learning**. Atlanta, GA, USA: JMLR.org, 2013. v. 28, iii–1310–iii–1318. Cited 1 time on page 22.

RAIMUNDO, Marcos M.; DRUMOND, Thalita F.; MARQUES, Alan Caio R.; LYRA, Christiano; ROCHA, Anderson; VON ZUBEN, Fernando J. Exploring multiobjective training in multiclass classification. **Neurocomputing**, v. 435, p. 307–320, 2021. Cited 7 times on pages 17, 18, 32, 41.

RAIMUNDO, Marcos M.; FERREIRA, P.; ZUBEN, F. V. An extension of the non-inferior set estimation algorithm for many objectives. **European Journal of Operational Research**, v. 284, p. 53–66, 2020. Cited 5 times on pages 35, 37, 62.

RAIMUNDO, Marcos Medeiros. **Multi-objective optimization in machine learning**. 2018. PhD thesis – University of Campinas - School of Electrical and Computer Engineering. DOI: 10.47749/T/UNICAMP.2018.1082001. Cited 1 time on page 32, 33, 35, 36.

REAL, Esteban; AGGARWAL, Alok; HUANG, Yanping; LE, Quoc. Regularized Evolution for Image Classifier Architecture Search. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 33, Feb. 2018. DOI: 10.1609/aaai.v33i01.33014780. Cited 1 time on page 25.

REAL, Esteban; MOORE, Sherry; SELLE, Andrew; SAXENA, Saurabh; SUEMATSU, Yutaka Leon; TAN, Jie; LE, Quoc V.; KURAKIN, Alexey. Large-Scale Evolution of Image Classifiers. In: ICML'17. **Proceedings of the 34th International Conference on Machine Learning**. [S.l.]: PMLR, Aug. 2017. v. 70, p. 2902–2911. Available from: <http://proceedings.mlr.press/v70/real17a.html>. Cited 2 times on pages 24, 25.

REN, Pengzhen; XIAO, Yun; CHANG, Xiaojun; HUANG, Po-yao; LI, Zhihui; CHEN, Xiaojiang; WANG, Xin. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 54, n. 4, May 2021. Cited 1 time on page 17, 26.

ROMERO, Carlos; REHMAN, Tahir. **Multiple criteria analysis for agricultural decisions**. [S.l.]: Elsevier, 2003. v. 11. Cited 1 time on page 35.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, n. 6, p. 386–408, 1958. ISSN 0033-295X. DOI: 10.1037/h0042519. Available from: <http://dx.doi.org/10.1037/h0042519>. Cited 1 time on page 19.

RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning Representations by Back-propagating Errors. **Nature**, v. 323, n. 6088, p. 533–536, 1986. DOI: 10.1038/323533a0. Available from: <http://www.nature.com/articles/323533a0>. Cited 1 time on page 20.

WIDROW, Bernard; HOFF, Marcian E. Adaptive Switching Circuits. In: WESTERN ELECTRONIC SHOW AND CONVENTION LOS ANGELES. **1960 IRE WESCON Convention Record, Part 4**. New York: IRE, 1960. P. 96–104. Cited 1 time on page 20.

WIECEK, Margaret; EHRGOTT, Matthias; ENGAU, Alexander. Continuous Multiobjective Programming. In: **Multiple Criteria Decision Analysis: State of the Art Surveys**. [S.l.: s.n.], Jan. 2016. v. 233, p. 739–815. DOI: 10.1007/978-1-4939-3094-4_18. Cited 1 time on page 32.

WISTUBA, Martin; RAWAT, Ambrish; PEDAPATI, Tejaswini. A Survey on Neural Architecture Search. **arXiv preprint arXiv:1905.01392**, 2019. arXiv: 1905.01392. Cited 7 times on pages 17, 24, 25, 26, 27.

XU, Yuhui; XIE, Lingxi; ZHANG, Xiaopeng; CHEN, Xin; QI, Guo-Jun; TIAN, Qi; XIONG, Hongkai. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In: ICLR 2020. **International Conference on Learning Representations**. [S.l.: s.n.], 2020. Available from: <https://openreview.net/forum?id=BJlS634tPr>. Cited 6 times on pages 18, 28, 30, 31, 44, 48, 60, 61.

YU, Kaicheng; SCIUTO, Christian; JAGGI, Martin; MUSAT, Claudiu; SALZMANN, Mathieu. Evaluating the Search Phase of Neural Architecture Search. **arXiv preprint arXiv:1902.08142**, 2019. arXiv: 1902.08142. Cited 1 time on page 25.

ZHOU, Zhi-Hua. **Ensemble methods: foundations and algorithms**. [S.l.]: CRC Press, 2012. Cited 1 time on page 33.

ZOPH, Barret; LE, Quoc V. Neural Architecture Search with Reinforcement Learning. In: ICLR 2017. **International Conference on Learning Representations**. [S.l.: s.n.], 2017. Available from: <https://openreview.net/forum?id=r1Ue8Hcxg>. Cited 5 times on pages 24, 25, 34.

ZOPH, Barret; VASUDEVAN, Vijay; SHLENS, Jonathon; LE, Quoc V. Learning Transferable Architectures for Scalable Image Recognition. In: CVPR 2018. **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], June 2018. Cited 1 time on page 24.