UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Rafael Claro Ito

OFA^2: A multi-objective perspective for the Once-for-All neural architecture search

OFA^2: Uma perspectiva multiobjetivo para a busca de arquitetura neural Once-for-All

Campinas
2023

Rafael Claro Ito

OFA²: A multi-objective perspective for the Once-for-All neural architecture search

OFA²: Uma perspectiva multiobjetivo para a busca de arquitetura neural Once-for-All

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, in 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 Área de Engenharia de Computação.

Supervisor: Prof. Dr. Fernando José Von Zuben

Este trabalho corresponde à versão final da dissertação defendida pelo aluno Rafael Claro Ito, e orientada pelo Prof. Dr. Fernando José Von Zuben.

Campinas
2023

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

It6o

Ito, Rafael Claro, 1992-
OFA²: A multi-objective perspective for the Once-for-All neural architecture search / Rafael Claro Ito. – Campinas, SP : [s.n.], 2023.

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

1. Busca de arquitetura neural (Aprendizado de máquina). 2. Otimização multiobjetivo. 3. Visão por computador. 4. Aprendizado de máquina. 5. Inteligência artificial. I. Von Zuben, Fernando José, 1968-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações Complementares

Título em outro idioma: OFA²: Uma perspectiva multiobjetivo para a busca de arquitetura neural Once-for-All
Palavras-chave em inglês:
Neural architecture search (Machine learning)
Multi-objective optimization
Computer vision
Machine learning
Artificial intelligence
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Fernando José Von Zuben [Orientador]
Marcos Medeiros Raimundo
Emely Pujólli da Silva
Data de defesa: 28-02-2023
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-1094-9191
- Currículo Lattes do autor: http://lattes.cnpq.br/3429903504683989

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato: Rafael Claro Ito RA: 118430
Data de defesa: 28 de fevereiro de 2023
Título da Tese em inglês: OFA^2: A multi-objective perspective for the Once-for-All neural architecture search
Título da Tese: OFA^2: Uma perspectiva multiobjetivo para a busca de arquitetura neural Once-for-All

Prof. Dr. Fernando José Von Zuben (FEEC/Unicamp)
Dra. Emely Pujólli da Silva (IC/Unicamp)
Prof. Dr. Marcos Medeiros Raimundo (IC/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.

This dissertation is dedicated to my wife Mariana for all the help and understanding during the period this project was developed. Without her support, I am sure this work would not have been completed the way it was. Gratitude.

Acknowledgements

Part of the results presented in this work were obtained through the project "Hub of Artificial Intelligence in Health and Wellbeing – Viva Bem", funded by Samsung Eletrônica da Amazônia Ltda.

"Education does not change the world. Education changes people. People transform the world."

(Paulo Freire)

Resumo

Projetar e treinar uma rede neural profunda requer alguma competência do usuário e geralmente consome um certo tempo. A área de AutoML visa automatizar este processo e começou a ganhar atenção nos últimos anos. Especificamente, a subárea de *Neural Architecture Search* (NAS) tenta resolver o problema de projetar e fazer a busca de uma rede neural dado um problema específico. Neste contexto, o *framework* de NAS denominado *Once-for-All* (OFA) abordou o problema de busca de arquiteturas eficientes para diferentes dispositivos com diversas restrições de recursos, desacoplando as etapas de treinamento e de busca. Isso significa que o custoso processo de treinamento de uma rede neural é feito apenas uma vez, sendo então possível realizar múltiplas buscas por sub-redes extraídas desta única super-rede, visando o atendimento de objetivos específicos. Neste trabalho uma nova estratégia de busca chamada OFA^2 é proposta, deixando a etapa de busca mais eficiente ao conceber explicitamente a busca como um problema de otimização multiobjetivo. Uma fronteira de Pareto é então povoada com arquiteturas neurais eficientes e já treinadas, exibindo diferentes compromissos entre os objetivos conflitantes. Isso pode ser alcançado usando um algoritmo evolutivo multiobjetivo durante a etapa de busca, como NSGA-II, SMS-EMOA ou SPEA2. Em outras palavras, a rede neural é treinada uma vez, a busca por sub-redes considerando diferentes restrições de *hardware* também é feita uma única vez, e então o usuário pode escolher uma rede neural adequada de acordo com cada cenário de implantação. A combinação do OFA com um algoritmo de otimização multiobjetivo explícito abre a possibilidade de tomada de decisão a posteriori em NAS, dado que um conjunto de sub-redes eficientes e já treinadas que aproximam a fronteira de Pareto são todas fornecidas de uma vez depois do estágio de busca. Além disso, também propomos uma seleção multiobjetivo denominada OFA^3, uma extensão da busca multiobjetivo OFA^2 que encontra automaticamente o número de participantes e os próprios modelos para formar ensembles eficientes. Este método de seleção leva a ensembles com melhor desempenho quando comparado a arquiteturas únicas. O código-fonte e o algoritmo de busca final podem ser encontrados em <https://github.com/ito-rafael/once-for-all-2>.

Palavras-chaves: Busca de arquitetura neural; *Once-for-All*; Otimização multi-objetivo; Tomada de decisão a posteriori.

Abstract

Designing and training a deep neural network requires some user expertise and is often time-consuming. The field of AutoML aims to automate this process and started gaining some attention in recent years. Specifically, the neural architecture search (NAS) subfield attempts to solve the problem of designing and searching a neural network given a specific problem. In this context, the Once-for-All (OFA) NAS framework addressed the problem of searching efficient architectures for different devices with various resource constraints by decoupling the training and the searching stages. This means that the costly process of training a neural network is done only once, and then it is possible to perform multiple searches for subnetworks extracted from this single super-network, focused on specific purposes. In this work a new search strategy called OFA^2 is proposed, making the search stage more efficient by explicitly conceiving the search as a multi-objective optimization problem. A Pareto frontier is then populated with efficient, and already trained, neural architectures exhibiting distinct trade-offs among the conflicting objectives. This could be achieved by using an evolutionary algorithm during the search stage, such as NSGA-II, SMS-EMOA or SPEA2. In other words, the neural network is trained once, the searching for subnetworks considering different hardware constraints is also done one single time, and then the user can choose a suitable neural network according to each deployment scenario. The combination of OFA and an explicit algorithm for multi-objective optimization opens the possibility of a posteriori decision-making in NAS, given that a set of efficient and already trained subnetworks that approximate the Pareto frontier are all provided at once after the search stage. Furthermore, we also propose a multi-objective selection called OFA^3, an extension of the OFA^2 multi-objective search which automatically finds the number of participants and the models themselves to form efficient ensembles. This selection method leads to ensembles with improved performance when compared to single architectures. The source code and the final search algorithm are released at <https://github.com/ito-rafael/once-for-all-2>.

Keywords: Neural Architecture Search; Once-for-All; Multi-objective Optimization; A posteriori decision-making.

List of Figures

List of Tables

List of Abbreviations

AutoML	Automated Machine Learning
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
EA	Evolutionary Algorithms
EMOA	Evolutionary Multiobjective Optimization Algorithm
GA	Genetic Algorithm
GPU	Graphics Processing Unit
HPO	Hyperparameter Optimization
ML	Machine Learning
MOO	Multi-objective Optimization
MOOP	Multi-objective Optimization Problems
NAS	Neural Architecture Search
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OFA	Once-for-All
RNN	Recurrent Neural Network
SMS-EMOA	S-Metric Selection Evolutionary Multiobjective Optimization Algorithm
SOTA	State-of-the-art
TPU	Tensor Processing Unit

Contents

1 Introduction

Artificial intelligence is far away from being a recent research area. However, this field gained a lot of popularity since 2012 with the introduction of the AlexNet neural network (Krizhevsky et al., 2012) during the ILSVRC-2012 ImageNet (Deng et al., 2009) competition. Since then, several other neural networks won this competition, year after year always pushing the overall scores higher. ZFNet (Zeiler & Fergus, 2014) in 2013, GoogLeNet (Szegedy et al., 2015) and VGG (Simonyan & Zisserman, 2015) in 2014, ResNet (He et al., 2016) in 2015, ResNeXt (Xie et al., 2017) in 2016, and finally SENet (Hu et al., 2018) in 2017, the last year of the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) competition. However, there is one important task inside the machine learning (ML) pipeline related to the modeling stage that is common to all of these neural networks cited previously, which can be very time-consuming: the design of these neural architectures were all hand-crafted by a research team. In this context, a recent research area called Neural Architecture Search (NAS) arose aiming to automate the process of designing a neural architecture, while improving the efficiency of the network search.

In this research field of NAS there is one framework called Once-for-All (OFA) (Cai et al., 2020) that decouples the training and the searching stages. This means that a large neural network is trained only once using a specific algorithm called *progressive-shrinking*, and then multiple searches can be done in the search space of subnetworks in order to find efficient architectures for different hardware constraints. This is a remarkable achievement, because now the search stage is very fast, given that any sampled subnetwork is already trained.

The work presented here extends the search stage of the Once-for-All network to make it even more efficient, by explicitly formulating and solving the NAS as a multi-objective optimization problem. This was accomplished with the help of an evolutionary multi-objective optimization algorithm (EMOA), exemplified by NSGA-II, SMS-EMOA and SPEA2. Although multi-objective solutions are more costly than single-objective ones due to the necessity of populating the Pareto frontier, here we are incrementing solely the cost of the search stage of the OFA framework, which is much faster than the training stage. Therefore, solving the search stage as a multi-objective optimization problem promotes a positive cost-benefit relation, opening the possibility of a posteriori decision-making based on the proposal of multiple efficient subnetworks exhibiting distinct trade-offs between two (or more) conflicting objectives (e.g. accuracy versus latency).

We list the main contributions of this work as follows:

- We propose a multi-objective perspective for the search stage of the Once-for-All framework and solve this multi-objective optimization problem by finding in a single search procedure multiple efficient architectures for different hardware constraints (e.g.: latency or FLOPS). Figure 1.1 summarizes the difference between the Once-for-All framework and our proposed method.
- We provide a series of experiments related to committee machines, and we empirically show that choosing efficient (also known as non-dominated) architectures to form an ensemble beats the performance of ensembles composed of random architectures.
- We propose a genetic algorithm that automatically selects different neural networks to form an ensemble. The ensembles found by the algorithm leads to more efficient final models (in terms of accuracy and latency) than the solutions obtained at the end of the search stage of the OFA framework.

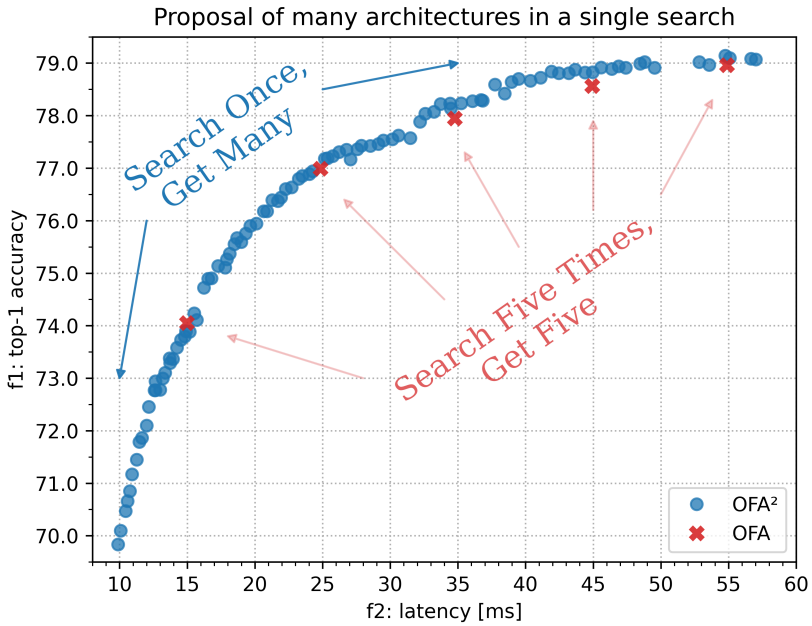


Figure 1.1 – Comparison between the search stage of OFA and OFA².

The next chapters are organized as follows: Chapter 2 summarizes the two main research areas related to this work, being Neural Architecture Search (NAS) and Multi-Objective Optimization (MOO), also mentioning related works involving both fields. Then, Chapter 3 will depict the main contributions of this work: after a brief explanation of the Once-for-All framework, details related to the multi-objective optimization formulation of the search stage will be given, and the key concepts relevant to the experiments with ensembles are discussed. All experiments and their respective results are reported in Chapter 4. Finally, Chapter 5 summarizes the overall contributions of this work.

2 Literature Review

This chapter summarizes the main concepts adopted in this work and some of their related works. Section 2.1 explains the idea behind the Neural Architecture Search (NAS) and its core components. Section 2.2 explains the fundamental concepts related to Multi-Objective Optimization (MOO), taking the content of Deb (2014) as the main reference.

2.1 Neural Architecture Search

Neural Architecture Search (NAS) is one of the three main subfields of the automated machine learning (AutoML) research area (Hutter et al., 2019). Besides NAS, there is another subfield of AutoML known as Hyperparameter Optimization (HPO). When a neural network is being trained, there are several hyperparameters that should be chosen and tuned to get the best results with the data provided. Examples of hyperparameters when training a neural network are learning rate, number of layers, number of neurons per layer and number of epochs to be trained. The last subfield of AutoML is Meta-Learning, which tries to reuse previous knowledge when training a neural architecture instead of doing it from scratch every time. Transfer learning, few-shot learning and zero-shot learning are examples of meta-learning. Figure 2.1 shows a non-exhaustive list of other research groups around the globe working with NAS.

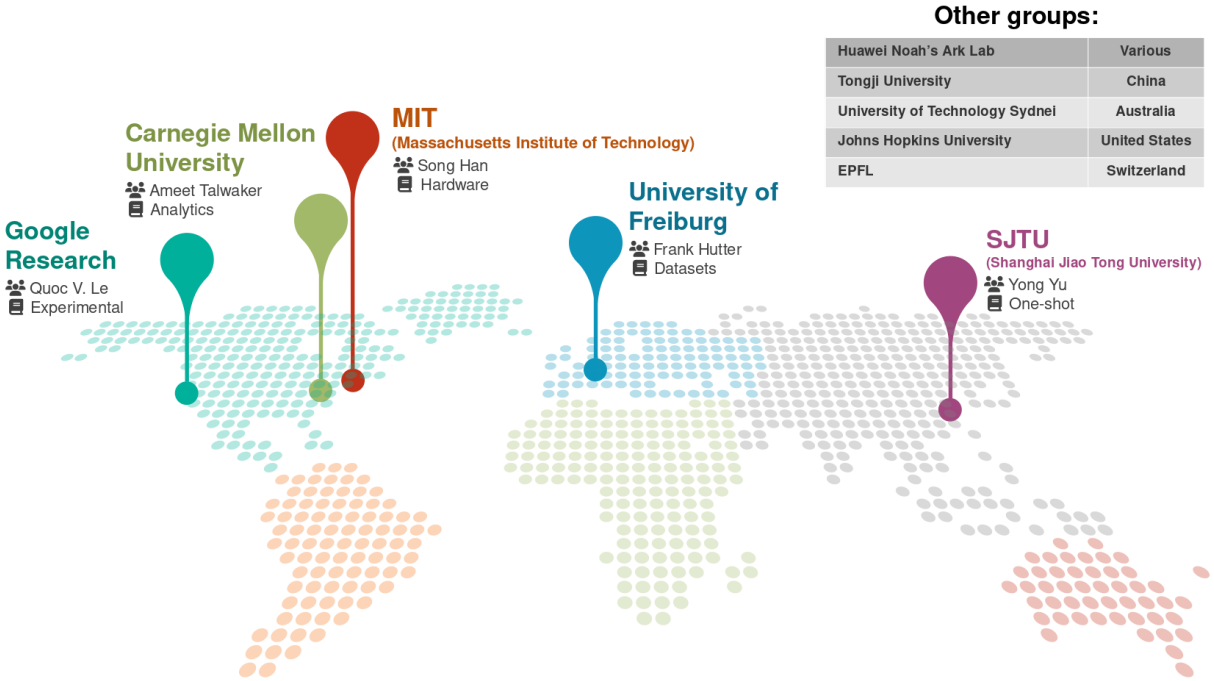


Figure 2.1 – Research groups working with NAS (period of search: 2020~2022).

We can associate each main subfield of AutoML with the problem it tries to solve by thinking about the whole pipeline of designing and training a neural network. While the goal of NAS techniques is to search the best architecture or set of architectures for a given problem, the HPO algorithms try to find the best hyperparameters values, and the meta-learning aims to incorporate prior knowledge for more efficient training. Figure 2.2 shows the mentioned main subfields of the AutoML research area.

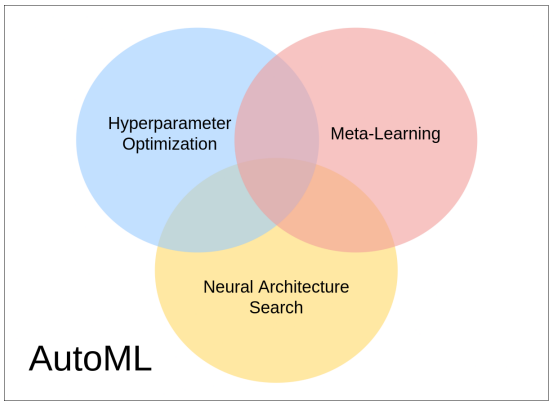


Figure 2.2 – Subfields of the automated machine learning (AutoML) research area.

The intersection areas in Figure 2.2 do not have any label associated because they do not have a specific title. It just means that more than one strategy is being used simultaneously to solve a particular problem. For example, consider the situation where a researcher is trying to solve a problem by using a ResNet (He et al., 2016) architecture with the weights of all layers frozen, except the last one, in order to perform transfer learning. Simultaneously, this researcher uses the Hyperband (Li et al., 2018) algorithm to tune the hyperparameters of the training, therefore using techniques from both meta-learning and HPO. Take another example where a researcher is using both a NAS framework to search the neural architecture and another hyperparameter optimization algorithm, which in this case would correspond to the intersection area between the HPO (in blue) and the NAS (in yellow) areas of Figure 2.2. From now on, the main focus will be on the NAS subfield.

There are usually three associated steps when implementing NAS, which are the definition of the search space, the search strategy, and the evaluation of the networks found (Elsken et al., 2019). Figure 2.3 illustrates these steps, to be properly presented in what follows.

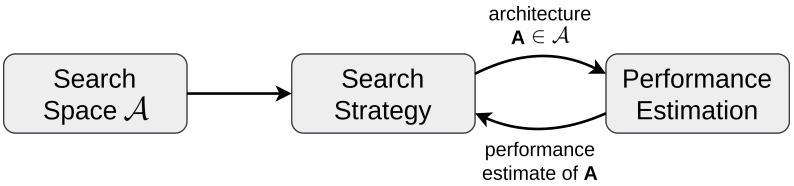


Figure 2.3 – Overview of the NAS framework. Adapted from Elsken et al. (2019).

2.1.1 Search Space

The search space defines the architectures that will be considered during the search. There are basically two types of search spaces available in NAS: the cell-based and the one-shot (Wistuba et al., 2019). In the cell-based formulation, the search is usually for a normal and a reduction cell that will be stacked later to form the final architecture. On the other hand, in the one-shot formulation, a super-network is trained to provide numerous internal subnetworks with shared weights.

Cell-based formulation

In the cell-based formulation, the solution for the NAS problem typically consists in finding two types of cells that will later be stacked in a specific manner to form the final architecture. The cell is represented by a DAG, in which nodes usually represent the operations and edges represent the flow of information. The two types of cells searched are the normal cell, that seeks to extract advanced features while keeping the spatial resolution untouched, and the reduction cell, with the main role of reducing the spatial resolution. Figure 2.4 shows an example of a cell represented as a directed acyclic graph (DAG). In this example the nodes are the operations and the edges simply show the flow of information (Ying et al., 2019). Figure 2.5 shows an example of how normal and reduction cells can be stacked to build the final architecture. Note that while the normal cell is stacked N times, the reduction cell is placed only once.

Zoph & Le (2017) adopted reinforcement learning as the searching strategy, Real et al. (2017) resorted to evolutionary algorithms and Xie et al. (2022) used a hybrid gradient-based approach, all three working with the cell-based search space.

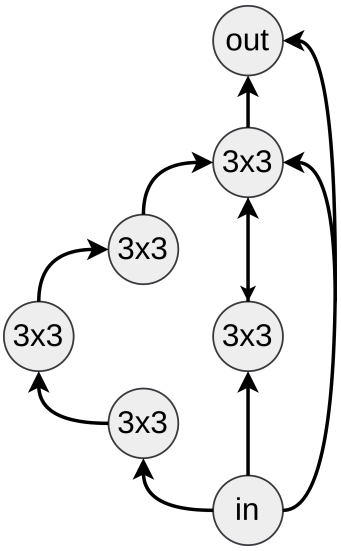


Figure 2.4 – Cell in the form of a DAG. Adapted from Ying et al. (2019).

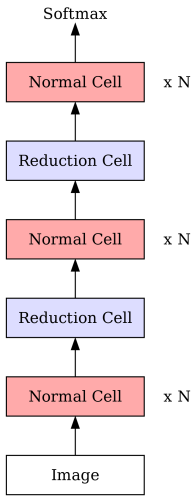


Figure 2.5 – Final architecture.

One-shot formulation

The one-shot formulation, also known as macro-level or global search, heavily relies on weight sharing among the architectures searched. In this kind of search space, training the weights for a particular neural network has an impact for other architectures as well. No particular examples will be provided here, since there are many techniques, each with its own particularities. Furthermore, the Once-for-All framework used in this work can be considered as a one-shot architecture, and it will be explained in details in Chapter 3.

Please check Bender et al. (2018), Brock et al. (2022) and Cai et al. (2019) for a review of NAS using the one-shot formulation.

2.1.2 Search Strategies

The search strategy defines how the search space will be explored, taking into account the exploration-exploitation dilemma. Several strategies can be used, such as random search (Li & Talwalkar, 2020), Bayesian optimization (White et al., 2021), encoder and decoder (Luo et al., 2018), and virtually any other strategy that may guide the search throughout the search space. Next, we will describe the most common ones, namely reinforcement learning, evolutionary algorithms and gradient-based approaches.

Reinforcement Learning

The first known successful attempt to actually fully train a neural network in an automated fashion was done in 2016, and it used reinforcement learning as the searching strategy (Zoph & Le, 2017). Figure 2.6 shows a general diagram of a reinforcement learning framework (Sutton & Barto, 2018).

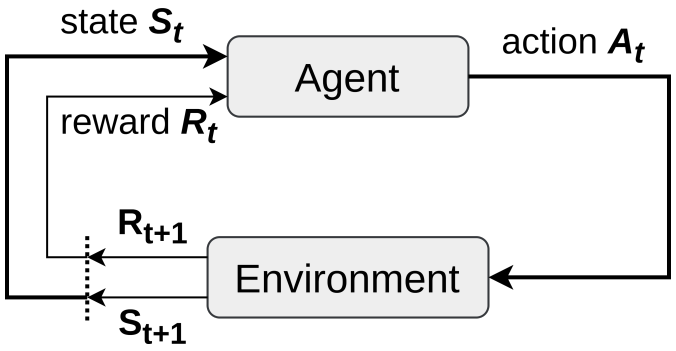


Figure 2.6 – General framework of a Reinforcement Learning algorithm. Adapted from Sutton & Barto (2018).

The controller was implemented as a recurrent neural network (RNN) and its role was to generate architectures. The reward was the performance of the sampled

architecture in the validation set, and the training aimed to maximize the expected accuracy in this data set. The final model achieved state-of-the-art (SOTA) results on the CIFAR-10 dataset (Krizhevsky, 2009), but the amount of GPUs (graphics processing units) needed to train this controller was huge, reported as 800 in the original paper.

Another similar and important work from the same authors introduced the NASNet search space (Zoph et al., 2018). The bottleneck of the technique, nevertheless, was the same: computational resources to perform the training of the controller. Recent works managed to overcome this burden, such as the TuNAS framework (Bender et al., 2020) introduced in 2020 that needs only a couple of hours in a single TPU (tensor processing unit) to search the architectures.

Evolutionary Algorithms

The second strategy to solve the NAS problem in deep learning used evolutionary algorithms, with the first work published in 2017 (Real et al., 2017). Figure 2.7 shows the general framework of an evolutionary algorithm.

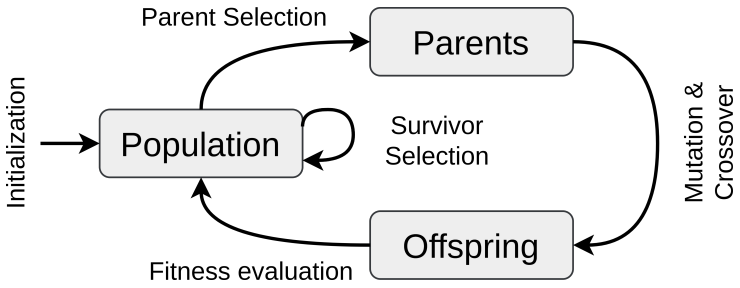
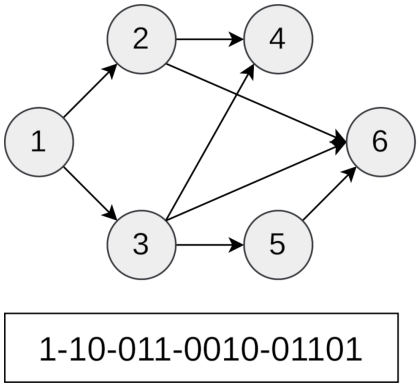


Figure 2.7 – General framework of an Evolutionary Algorithm. Adapted from Wistuba et al. (2019).

Evolutionary algorithms (EAs) are often adopted when looking at the formulation of NAS problems with a multi-objective optimization perspective. Section 2.2 will cover the topic of multi-objective optimization and explain the core components present in a general evolutionary algorithm, as the ones displayed in the diagram of Figure 2.7.

A common approach to solve a NAS problem with EAs is to search for a directed acyclic graph (DAG), also called a *cell* in the context of NAS. Usually, the nodes of this DAG represent the feature maps and the edges represent the operations done to the feature maps (convolutions, skip connection, and so on), although this is not a strict rule. This DAG is then encoded into a sequence of strings that is actually used by the algorithm. Figure 2.8 shows an example of encoding used to represent a DAG (Lu et al., 2020).



	1	2	3	4	5	6
1	0	1	1	0	0	0
2	-1	0	0	1	0	1
3	-1	0	0	1	1	1
4	0	-1	-1	0	0	0
5	0	0	-1	0	0	1
6	0	-1	-1	0	-1	0

Figure 2.8 – Encoding of a cell. Adapted from Lu et al. (2020).

There are other works that use EAs to solve NAS, e.g. the AmoebaNet (Real et al., 2019).

Gradient-based search

Another acclaimed search strategy that popularized in recent years is based on gradient descent algorithms. This is very convenient because it allows the search of a neural architecture to be done by the same backpropagation method used to train the weights of the neural network (Rumelhart et al., 1986).

This class of search strategy usually defines architecture parameters α as well, besides the common weights w of the neural network. An example of training procedure consists in freezing the architecture parameters α while training the networks weights w on the training dataset, and then freezing the network weights w while training the architecture parameters α on the validation dataset. This procedure is repeated alternately. Figure 2.9 illustrates the framework of this strategy.

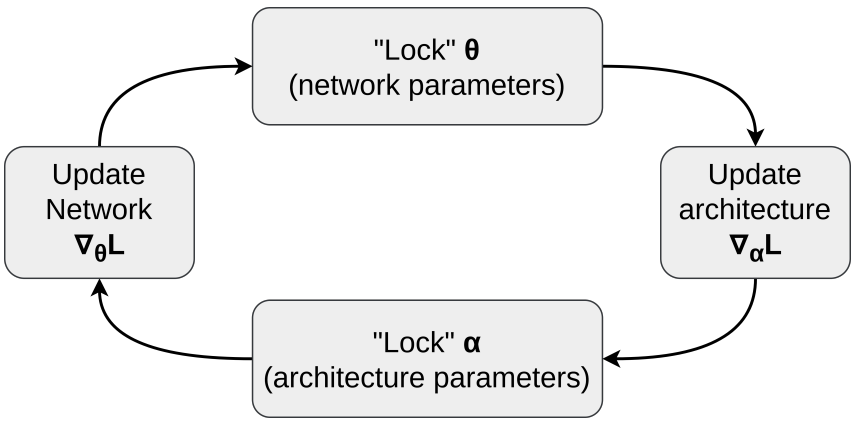


Figure 2.9 – Gradient-based search applied to NAS. Adapted from Green et al. (2019).

A popular algorithm using gradient descent to search the architecture is the DARTS framework (Liu et al., 2018). The key idea is to find a way on how to transform the combinatorial choice of discrete operations into a differentiable function, in order to make the search space continuous and suitable to be trained with the backpropagation procedure. This was achieved by applying a softmax with respect to the architecture parameters α, therefore considering all operations jointly, instead of a single operation at a time. Due to the softmax function characteristics, the architecture parameters α can now be interpreted as a probability. Then, a simple progressive pruning take the least probable α and deletes it, meaning that the operation associated to that parameter is now discarded. This is repeated until a single α parameter, and therefore a single operation, lasts. Figure 2.10 shows an overview of the DARTS framework (Zhu et al., 2021).

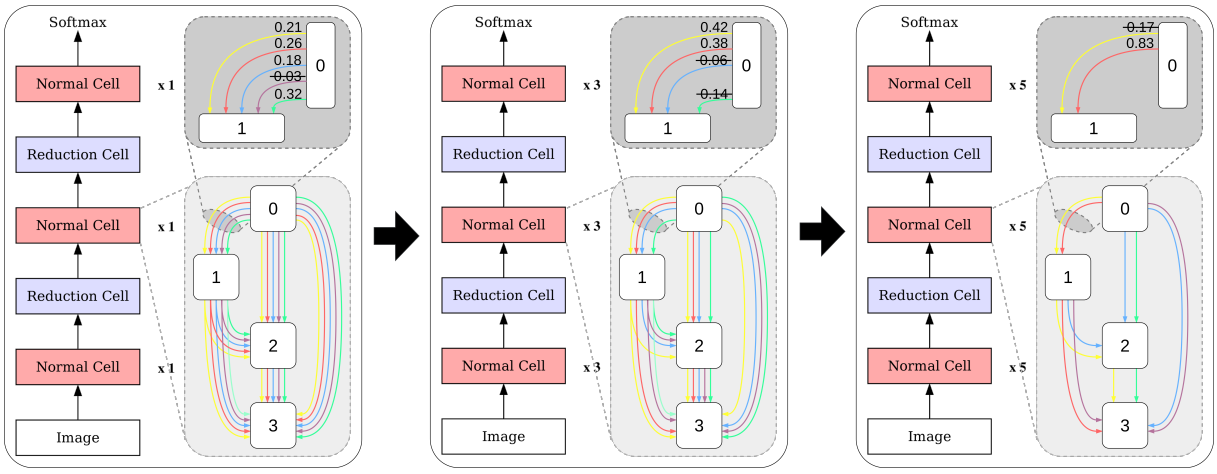


Figure 2.10 – Overview of the DARTS framework. Adapted from Zhu et al. (2021).

Several works adopted this idea and proposed extensions to the DARTS framework. P-DARTS (Chen et al., 2019), PC-DARTS (Xu et al., 2020), RAPDARTS (Green et al., 2019), DARTS+ (Liang et al., 2019) and I-DARTS (Jiang et al., 2019) are some of these extensions.

2.1.3 Evaluation

The performance estimation strategy is used to evaluate the architectures proposed by the search strategy. Early methods used to fully and independently train all networks searched during the optimization. This led to huge amount of resources. Later, strategies based on performance estimation were developed to speed up the evaluation of the architectures. Early stopping, using a proxy model, extrapolating the learning curve and network morphisms are just a few examples of attempts to estimate the network performance and guide the search more efficiently (Elsken et al., 2019).

2.2 Multi-objective Optimization

Multi-objective optimization (MOO) is an area concerned with solving problems when more than one objective function is being considered and need to be solved simultaneously. Ideally, these objectives are modeled as conflicting with each other in multi-objective optimization problems (MOOP).

A naive attempt to solve this class of problems is by scalarizing the multiple objectives into a single objective, and then solving the problem considering only this single goal. This is often treated as the classical approach for solving MOOP. There are, however, more suitable approaches that try to solve MOOP as it is. That is, techniques that consider the objective functions independently, without the need to create a single joint objective. Evolutionary algorithms are a good example of this type of optimizer and are often adopted to solve MOOP problems. In this respect, one of the challenges in MOO is to search not for a unique solution, but instead multiple efficient solutions characterized by distinct trade-offs of the objectives (known as *Pareto-optimal* solutions), each of them weighting the objective functions differently.

Let us consider the decision-making involved in buying an automobile car. Assume for this example that the cost and the comfort are chosen as the objective functions. The goal is to minimize the cost spent on the car, while maximizing its comfort. Note that the nature between these two objectives are inherently conflicting with each other: while a cheap car, although not providing you with the best comfort, can save you some money, the most comfortable automobiles on the market will certainly cost a fortune. So there is no single solution for this problem, but instead a set of efficient solutions with distinct trade-offs of the objective functions. Figure 2.11 illustrates a set of solutions exhibiting distinct trade-offs of the conflicting objective functions. Point 1 of the curve represents the solution that benefits more the cost, while Point 2 benefits more the comfort. Points A, B and C represent solutions characterized by distinct relative importance of the objective functions (cost and comfort).

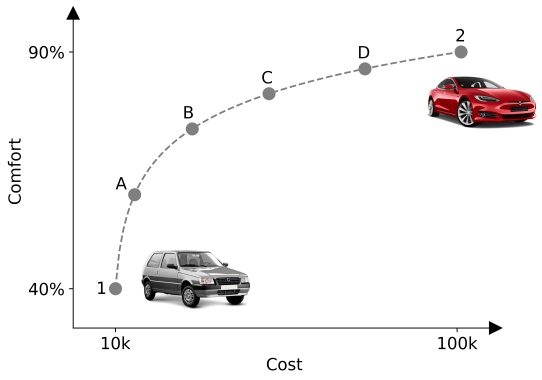


Figure 2.11 – Solutions with different trade-offs of the objectives. Adapted from Deb (2014).

2.2.1 Approaches to Multi-Objective Optimization

The optimization problems are typically associated with one of the next three classes:

- Single-objective optimization: only one single objective function is considered.
- Multi-objective optimization: two, three or four objective functions are considered.
- Many-objective optimization: more than four objective functions are considered.

The focus of this work is in multi-objective optimization considering two objective functions. In what follows, two different approaches for solving multi-objective optimization problem are compared.

A posteriori decision-making approach

The first approach discussed here is considered to be the ideal multi-objective optimization procedure. In this approach the first step after defining the problem is to search multiple efficient solutions exhibiting distinct trade-offs of all objective functions. Then, with a diverse set of optimal solutions in hand, the user have the power to choose a specific solution. The key point here is that, in this approach, the set of solutions are found first, and then the user choice comes a posteriori. Figure 2.12 shows a diagram with the steps of an ideal optimizer for multi-objective optimization problems.

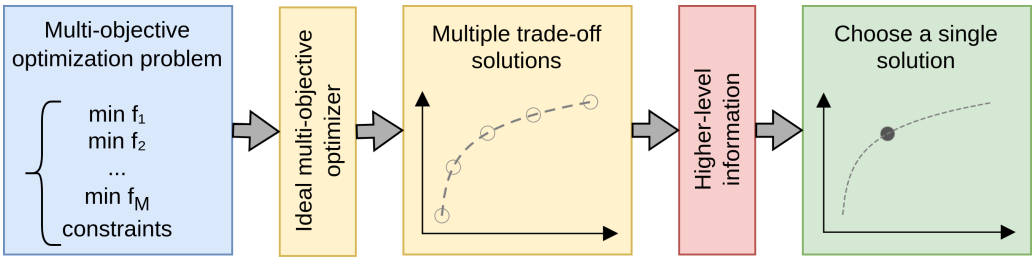


Figure 2.12 – Schematic of an ideal multi-objective optimization procedure. Adapted from Deb (2014).

A priori decision-making approach

The second approach is also known as the preference-based multi-objective optimization procedure. In this approach, the user choice enters straight after a proper definition of the problem and the specification of the multiple objective functions. In this stage, the user must define a degree of preference for one or more objectives, and then the optimization procedure can proceed. The difference is that now the optimization procedure should return a single solution satisfying the user input, if it exists. In this method it is said that the user choice comes a priori with respect to search stage. Figure 2.13 shows a diagram with the steps of a preference-based optimizer for MOOP.

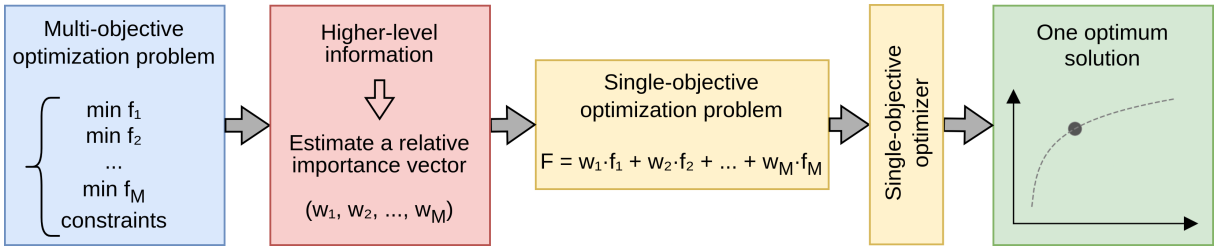


Figure 2.13 – Schematic of a preference-based multi-objective optimization procedure. Adapted from Deb (2014).

2.2.2 Decision Space and Objective Space

When dealing with single-objective optimization, the decision variable space is the only dimensional space involved in the problem, with the dimension of this space being equal to the number of decision variables. However, when multi-objective optimization is taken into consideration, the objective functions related to the problem also constitute a multidimensional space. The dimension of this space, called the objective space, is equal to the number of objective functions being considered in the optimization. Furthermore, for each candidate solution in the decision space there exists a respective point in the objective space. This mapping occurs from the decision space, containing all possible solutions to the problem, to the objective space, that evaluates these solutions on each of the objective functions. This mapping also implicitly occurs for single-objective optimization. But in this case, instead of involving spaces of two or more dimensions, it happens from the decision space to a single value, for example a real number. Figure 2.14 shows an example of the mapping between a three-dimensional decision space and a two-dimensional objective space.

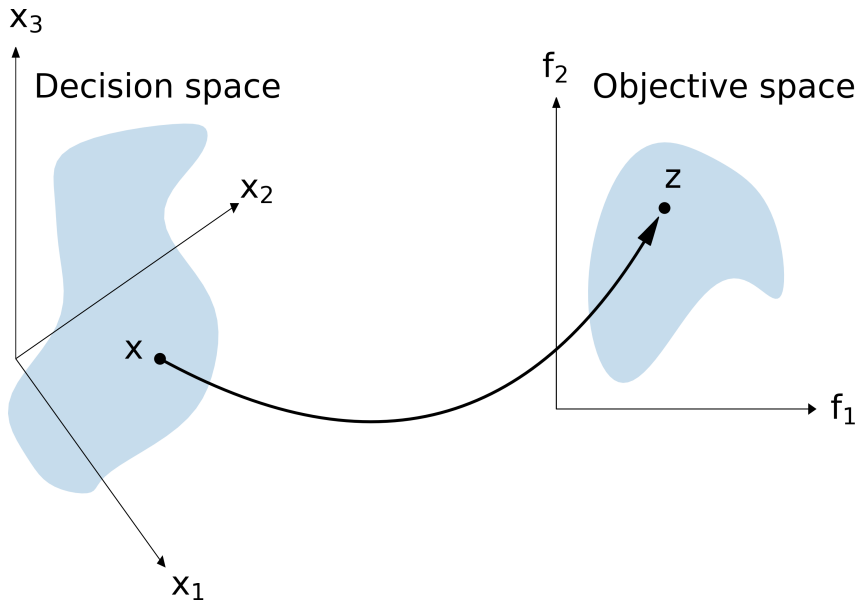


Figure 2.14 – An illustrative representation of the decision and the objective spaces. Adapted from Deb (2014).

2.2.3 Dominance

The concept of dominance is of high relevance when solving a multi-objective optimization problem, since it allows the evaluation on how good a candidate solution is, which can help to guide the search towards promising directions. It arises from the idea that given any two solutions, it would be convenient to be able to compare them somehow. The definition of dominance involves the objective space and can be stated as:

Definition 1 A solution x_1 is said to dominate another solution x_2 if both conditions are satisfied:

I. The solution x_1 is no worse than solution x_2 in all objectives.

II. The solution x_1 is strictly better than solution x_2 in at least one objective.

If either of these conditions is violated, the solution x_1 does not dominate the solution x_2. When solution x_1 dominates solution x_2, all the next statements are equivalent: x_2 is dominated by x_1; x_1 is non-dominated by x_2; x_1 is non-inferior to x_2. To illustrate this concept, Figure 2.15 brings an objective space with f_1 and f_2 as the objective functions. The problem consists in maximizing f_1 while minimizing f_2, and brings five solutions numerated from 1 to 5.

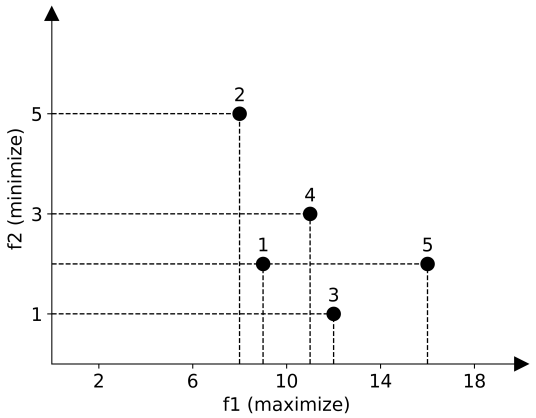


Figure 2.15 – MOOP with two objective functions displaying five candidate solutions. Adapted from Deb (2014).

According to Definition 1, a comparison between any two solutions can be done aiming at determining which solution is better when considering both objectives. For example, by comparing candidate solutions 2 and 4, it is possible to state that solution 4 dominates solution 2, since solution 4 is better than solution 2 in both f_1 and f_2 objectives. On the other hand, the same relationship can not be associated with solutions 3 and 5 (or equivalently with solutions 1 and 4). While solution 3 is better than solution 5 considering the objective function f_2, the opposite is true when considering the objective function f_1. In this case, neither of the solutions dominates the other.

2.2.4 Pareto frontier

Considering the concept of dominance presented in Definition 1, the *non-dominated set* is defined as follows:

Definition 2 Let P be a given set of candidate solutions. The non-dominated set of candidate solutions P' is a subset of P composed solely of non-dominated solutions, more specifically, the subset of candidate solutions that are not dominated by any other element of the set P.

In other words, given a finite set of candidate solutions, we can perform all possible pairwise comparisons to determine the non-dominated set of solutions. Therefore, for each candidate solution that does not belong to the non-dominated set, there is at least one from the non-dominated set that dominates that specific candidate solution, in the sense of not being worse in any objective and being better in at least one objective. Moreover, the comparison of two solutions belonging to the non-dominated set is not that assertive, because each of them is better than the other with respect to at least one objective function.

Still relying on Definition 2, when the set P comprises the entire search space, then the set of non-dominated solutions P' is called the *Pareto-optimal set*. The region of the objective space in which the Pareto-optimal set of solutions is located is called the Pareto front or Pareto frontier. Figure 2.16 highlights the Pareto front in the objective space considering all combinations of optimization problems for two objective functions.

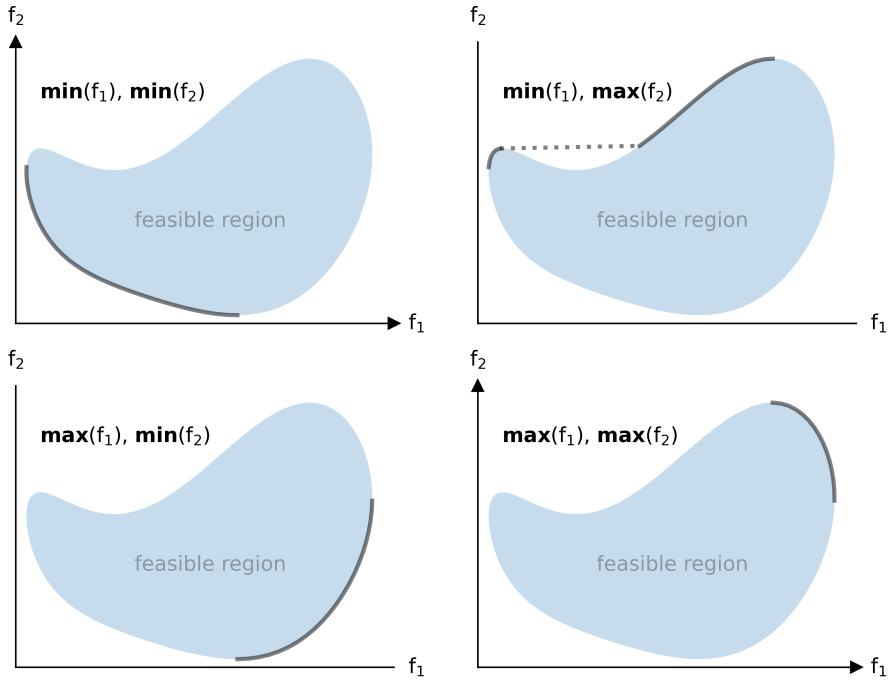


Figure 2.16 – Examples of Pareto-optimal solutions, given the set of all candidate solutions (gray area). Adapted from Deb (2014).

It is clear now that the ideal final answer for an MOOP is to find the Pareto-optimal set of solutions. However, this may not be an easy task to accomplish depending on the problem, as one might guess. The last key concept relevant to fully understand the algorithm used in this work is driven by the idea of grouping and classifying solutions in different levels of non-dominated fronts. This is almost like an extension of the previous concept of dominance. But instead of comparing two individual solutions, the solutions are now grouped in non-dominated fronts of different levels, with lower levels containing better solutions than higher levels.

To make it more clear, let us bring back the five solutions used in the example of Figure 2.15. By the aforementioned pairwise comparison involving all the candidate solutions, it can be stated that solutions 3 and 5 form the first non-dominated set, called the level 1 non-dominated front. Now, excluding the solutions of the level 1 front, the same dominance comparison can be done again. This ends up with solutions 1 and 4 being the non-dominating set among the remaining solutions. This set containing the solutions 1 and 4 is said to be the level 2 front of non-dominated solutions. Finally, the solution 2 is categorized in the set of the non-dominated solutions of level 3. Figure 2.17 illustrates all three levels of non-dominated fronts considering the same solutions provided by the example in Figure 2.15.

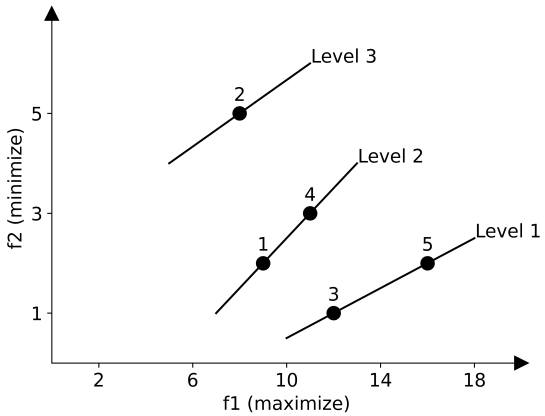


Figure 2.17 – Solutions grouped in different levels of non-dominated fronts. Adapted from Deb (2014).

Since evolutionary algorithms are often adopted to solve MOOP and given that they are founded on an iterative population-based search, having a way to group and compare different non-dominated fronts can be useful to guide the search towards finding better solutions. This sorting benefits the algorithm by bringing candidate solutions closer to the Pareto front, and it is indeed explored in several evolutionary methods.

3 Methodology

Our work is an extension of the Once-for-All framework that aims to improve the search stage by selecting diverse architectures with better performance, all at once. Since the training stage represents the step that demands most of the computational resources, we decided to take advantage of the fact that the OFA super-network was already trained and has its weights publicly available, and we focused on improving the search stage. More specifically, we formulated the search stage as a multi-objective optimization problem and solved it making the search more robust and leading to more efficient final architectures. Section 3.1 provides details related to the training procedure of the OFA super-network and Section 3.2 explains the multi-objective optimization problem proposed in this work. Lately, we use the set of solutions obtained from the search stage to form ensembles and compare them under different scenarios. Information related to ensembles are given in Section 3.3.

3.1 Training Stage

The Once-for-All (OFA) network (Cai et al., 2020) is a convolutional neural network (CNN) with the same architecture space of MobileNetV3 (Howard et al., 2019) trained as a one-shot NAS framework in which the training and the search stages are decoupled. That is, the network is trained only once, and then the search for a subnetwork inside this super-network is done without the need for additional training. Figure 3.1 illustrates this behavior.

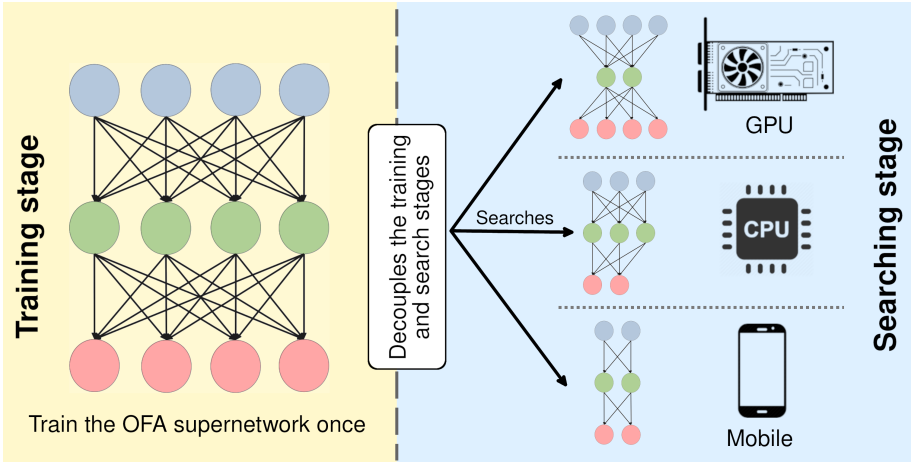


Figure 3.1 – Once-for-All framework.

Being able to train the neural network only once and then perform multiple searches is a huge advantage over the usual machine learning workflow, which requires

training one neural network for each deployment scenario. The OFA network can save computational resources when multiple hardware architectures are considered, exploiting the fact that the cost of searching for an architecture in the Once-for-All network is much cheaper than training a neural network from scratch. Figure 3.2 shows a comparison when searching four architectures with different hardware constraints between the OFA network (no need for retraining) and a MobileNetV3 (one training for each architecture searched) (Howard et al., 2019).

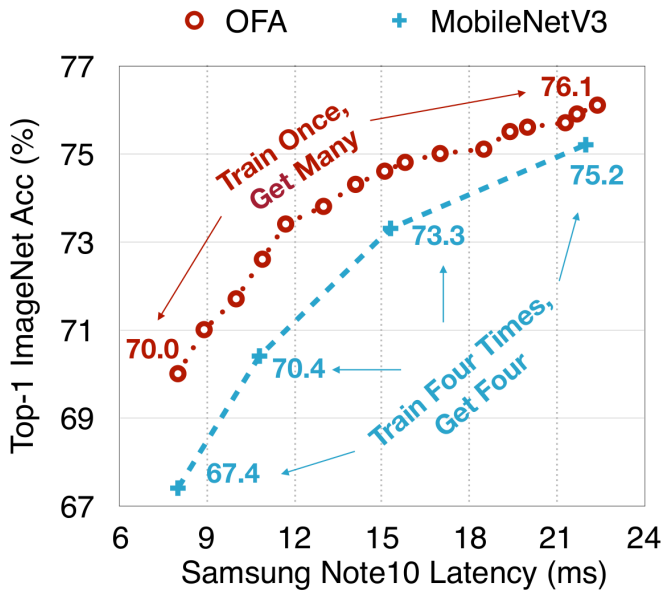


Figure 3.2 – MobileNetV3 and OFA training stage comparison. Figure from Cai et al. (2020).

Next we describe the architecture space and the procedures for the training stage of the OFA framework.

3.1.1 Progressive shrinking

The OFA network training is achieved by the *progressive shrinking* algorithm (Cai et al., 2020). The progressive shrinking (PS) scheme can be viewed as a generalized network pruning technique, but applied to more dimensions. The pruning technique usually shrinks the width of a neural network, decreasing the number of channels in a given layer, for example. Figure 3.3 shows a usual pruning procedure.

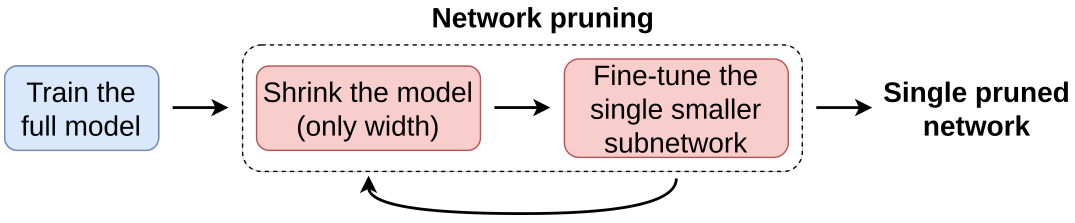


Figure 3.3 – Network pruning. Adapted from Cai et al. (2020).

The progressive shrinking scheme is similar to the pruning procedure, but instead of shrinking the model in only one dimension, it aims to shrink the model in four dimensions. Figure 3.4 shows the PS scheme. Next, we describe each of the dimensions shrinked by the PS algorithm, being: resolution, kernel size, depth and width.

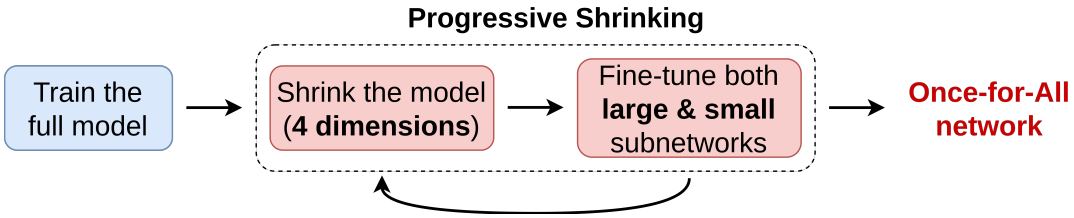


Figure 3.4 – Progressive shrinking. Adapted from Cai et al. (2020).

The training procedure starts by training the largest network possible and then progressively fine-tuning the network to nest smaller subnetworks inside larger subnetworks, shrinking one dimension at a time following the order shown in Figure 3.5. Starting training the neural network at its full capacity prevents the smaller subnetworks from interfering with the larger subnetworks.

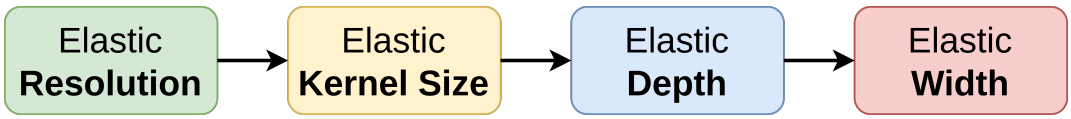


Figure 3.5 – Dimensions shrinked by the progressive shrinking process.

Elastic Resolution

Throughout the training, arbitrary image sizes are used as input for the convolutional neural network. This is denoted as *elastic resolution*. The input image size ranges from 128 up to 224 pixels with a stride of 4. Therefore, 25 different input resolutions are allowed (128, 132, . . . , 224). Figure 3.6 illustrates the elastic resolution.

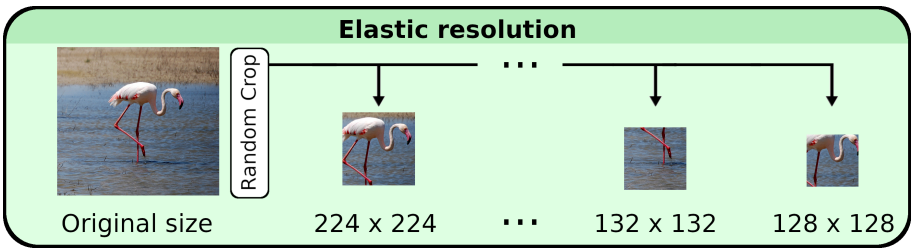


Figure 3.6 – Progressive Shrinking: Elastic Resolution.

Elastic Kernel Size

The *elastic kernel size* is related to the convolutional operations of the neural network. It is achieved by placing a 3×3 kernel inside a 5×5 kernel, which in turn is placed inside a

7×7 kernel. However, simply reusing the exact same elements might lead to performance degradation for some subnetworks, because the centered sub-kernels (3×3 and 5×5) are shared and can play multiple and different roles inside a kernel convolution. So instead of keeping them the same, a kernel transformation is done in order to obtain the smaller kernel sizes. That is, after training the Once-for-All network in its full size (with 7×7 kernel sizes), the 5×5 kernel can be obtained by taking the inner 5×5 elements inside the 7×7 kernel and multiplying them by a transformation matrix. The same is done with the inner 3×3 elements of the 5×5 kernel in order to obtain the 3×3 kernel. Figure 3.7 illustrates the elastic kernel size with the kernel transformation matrices.

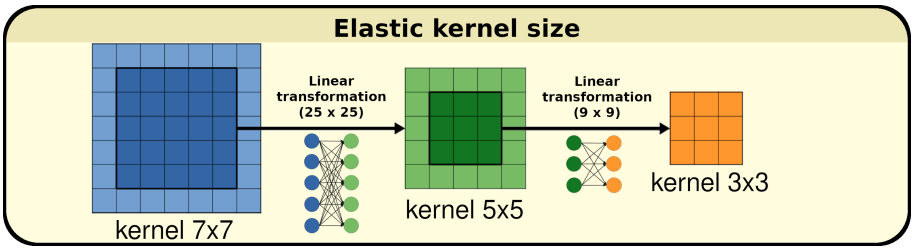


Figure 3.7 – Progressive Shrinking: Elastic Kernel Size.

Within each layer, these kernel transformation matrices are shared among different channels. For different layers, different matrices are used instead. The overhead for storing these kernel transformation parameters are of only $25 \times 25 + 9 \times 9 = 706$ extra parameters in each layer, which can be considered as negligible.

Elastic Depth

In order to sample a subnetwork with fewer layers than the full OFA network, the last layers are simply skipped one by one. First, the last layer is skipped and the remaining network is fine-tuned. Then, the last two layers are skipped and the remaining network is fine-tuned, and so on. This means that the weights of the first layers are shared between large and small architectures, while the weights of the last layers are used only by large networks. The elastic depth procedure occurs per unit and is shown in Figure 3.8. Each unit is allowed to have a depth of 2, 3 or 4 layers.

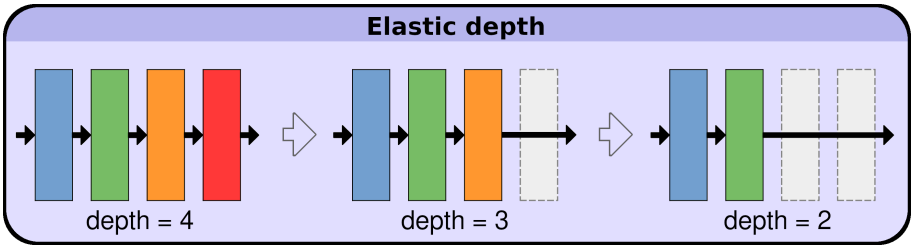


Figure 3.8 – Progressive Shrinking: Elastic Depth.

Elastic Width

The elastic width allows each unit to choose different channel expansion ratios. After training the OFA network with the maximum width, the progressive shrinking scheme sorts the channels according to their importance, and then skip the least important channels one at a time. The importance of each channel was calculated using the L1 norm of the channel's weights, with larger L1 norm meaning more importance. This way, when a smaller subnetwork is sampled, it starts with the most important channels. Figure 3.9 shows this procedure. Each unit is allowed to choose a channel expansion ratio of 3, 4 or 6.

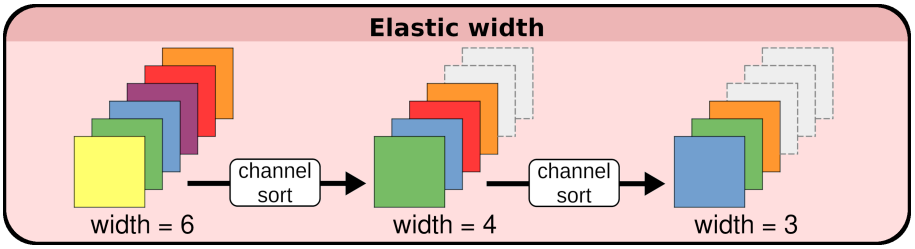


Figure 3.9 – Progressive Shrinking: Elastic Width.

Training details

The OFA network at its full capacity was trained using the SGD optimizer with Nesterov momentum 0.9 and weight decay $3e^{-5}$. The initial learning rate is 2.6, with cosine schedule for learning weight decay. The network was then trained for 180 epochs with batch size 2048 on 32 GPUs, taking around 1,200 GPU hours on V100 GPUs (Cai et al., 2020). The dataset used for both the training and searching stages is the ImageNet (Deng et al., 2009), a standard dataset in the computer vision area that consists of 1,281,167 images in the training set and 50,000 images in the validation set, organized in 1,000 categories.

In summary, each component of the progressive shrinking can be chosen according to what is shown in Table 3.1.

Table 3.1 – Number of available choices for each dimension of Progressive Shrinking

PS	property	choices	number of choices
elastic resolution	input resolution	{224, 220, . . . , 128}	25
elastic kernel	kernel size	{3, 5, 7}	3
elastic depth	layers	{2, 3, 4}	3
elastic width	channels	{3, 4, 6}	3

3.2 Search Stage

The Once-for-All network is one single convolutional neural network, but with plenty of subnetworks with different sizes nested inside of it. The architecture is composed of five units. Each unit can have a depth chosen from {2, 3, 4} layers, a width expansion ratio chosen from {3, 4, 6} and a kernel size chosen from {3, 5, 7}. This gives a search space of $((3 \times 3)^2 + (3 \times 3)^3 + (3 \times 3)^4)^5 \approx 2 \times 10^{19}$ different neural network architectures. Since all these subnetworks share the same weights, only 7.7 M parameters need to be stored. During the training, the input image size ranges from 128 to 224 with a stride of 4.

The original framework performs an evolutionary search on the trained OFA network to find an architecture with the meeting requirements based on restrictions imposed by the user. This means that the resource constraints must be defined a priori to the search process and that the output of the framework is a single architecture.

On the other hand, in our work, we propose the search process in a multi-objective perspective (Burke & Kendall, 2014), where the goal is to minimize two conflicting objective functions, more specifically the top-1 error and a hardware constraint (latency or FLOPS). We tested three multi-objective optimization (MOO) algorithms in our experiments, all being evolutionary and population-based: the NSGA-II (Deb et al., 2002), which uses the concepts of non-dominated sorting and crowding distance to solve the problem, the SMS-EMOA (Beume et al., 2007) which uses the hypervolume metric to perform the evolution, besides the non-dominated sorting concept as well, and the SPEA2 (Zitzler et al., 2001) characterized by using a fine-grained fitness assignment strategy based on a dominance index and a density index. Figure 3.10 illustrates this multi-objective search, which we call as OFA² search.

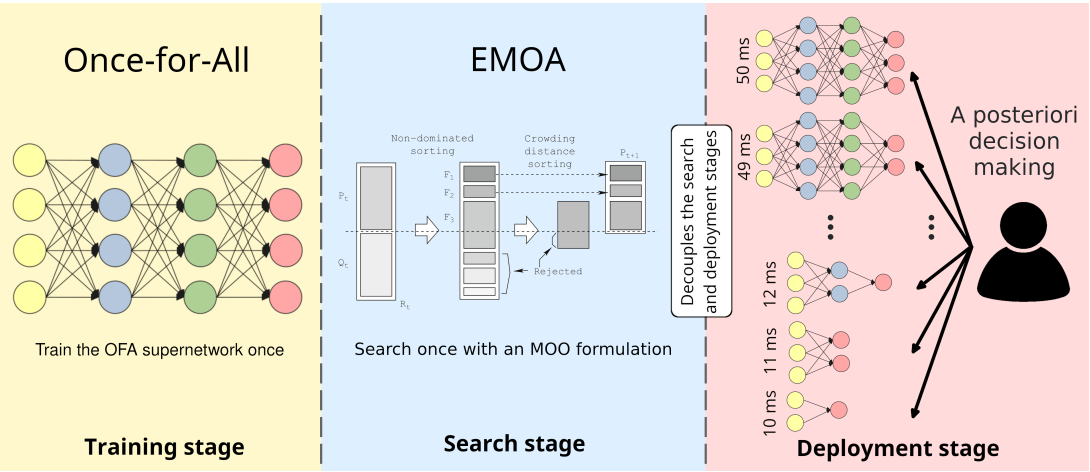


Figure 3.10 – OFA² search overview: the search and deployment stages are decoupled and the decision-making is now a posteriori, after the optimization process.

The whole problem was modeled using a multi-objective optimization framework called pymoo (Blank & Deb, 2020), which uses the NumPy (Harris et al., 2020) library as

its backend. For the evaluation of the neural network architectures, an NVIDIA GeForce GTX Titan X GPU with 12 GB of memory was used, with the code being implemented using the PyTorch framework (Paszke et al., 2019).

3.2.1 Genotype

The first step to solve the search stage as an MOO problem is to define the genotype encoding (also known as chromosome representation) that will be used by the evolutionary multi-objective algorithm (EMOA) to codify an individual of the population. Each individual represents a subnetwork and the population is a set of subnetworks considered at each iteration of the algorithm. For the genotype, we simply flatten the hyperparameters used to represent an architecture from Table 3.1 in a one dimensional array. Since the final architecture is formed by 5 units in sequence and each unit may have up to 4 layers, we have a total of 20 genes to represent the convolutional kernel size (`ks`) and 20 genes to represent the width (`w`). We also have 5 genes related to the depth (`d`) of each unit showing the number of layers considered, and one last gene informing the resolution (`r`) of the image that will be cropped from the dataset and used as the input of the neural network. This gives us a total of 46 variables for the encoding of each individual that represents a full neural network. Figure 3.11 illustrates this representation of an individual.

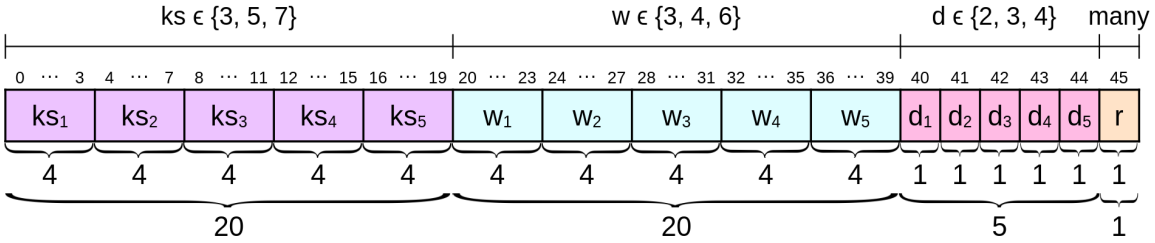


Figure 3.11 – Genotype encoding used to represent an individual of the population.

Given the encoding of an individual, in order to build the full architecture associated with that individual, the first step is to separate each variable according to the hyperparameter it represents, following the scheme depicted in Figure 3.11. Figure 3.12 illustrates a numerical example of an individual and Figure 3.13 shows the respective division of its values according to the hyperparameters associated. After that, we can group these hyperparameters per unit. If the depth of the unit being considered is equal to 4, i.e. the unit has 4 layers, then all 4 values of the kernel size and all 4 values of the width are valid and will be used to build the model. If the depth of the unit is equal to 3 layers, then we simply discard the last entry of both the kernel size and width. If the depth is equal to 2, then we discard the last 2 entries of both kernel size and width. Figure 3.14 illustrates this procedure. Finally, with all hyperparameters of the 5 units, we can build the full architecture, as shown in Figure 3.15.

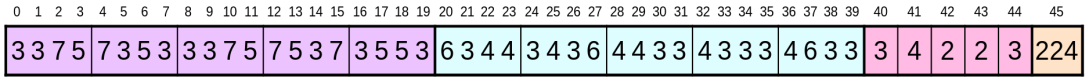


Figure 3.12 – Numerical example of an individual.

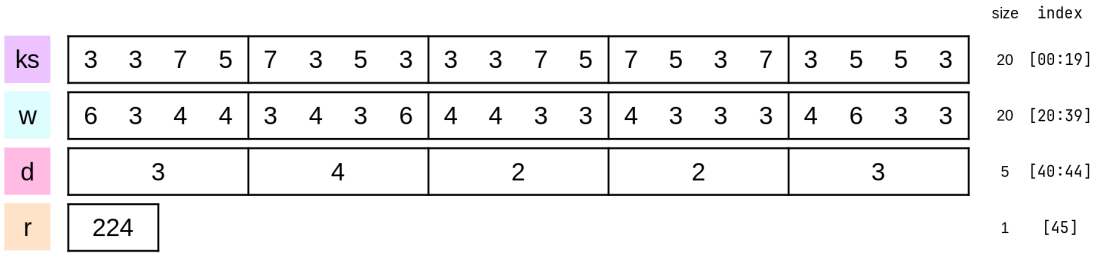


Figure 3.13 – Splitting the encoding according to each hyperparameter.

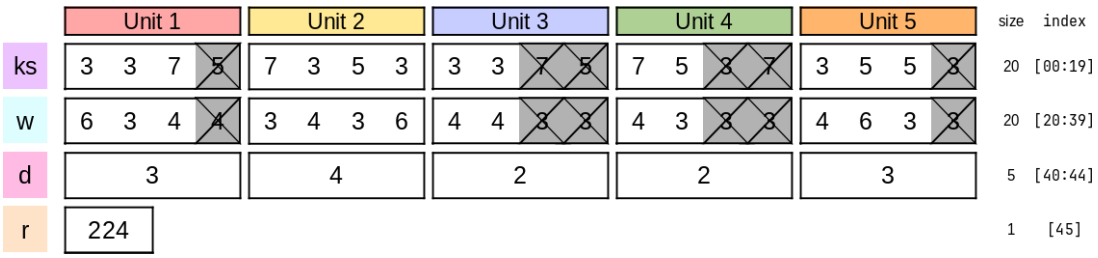


Figure 3.14 – Discarding entries when depth $\in \{2,3\}$.

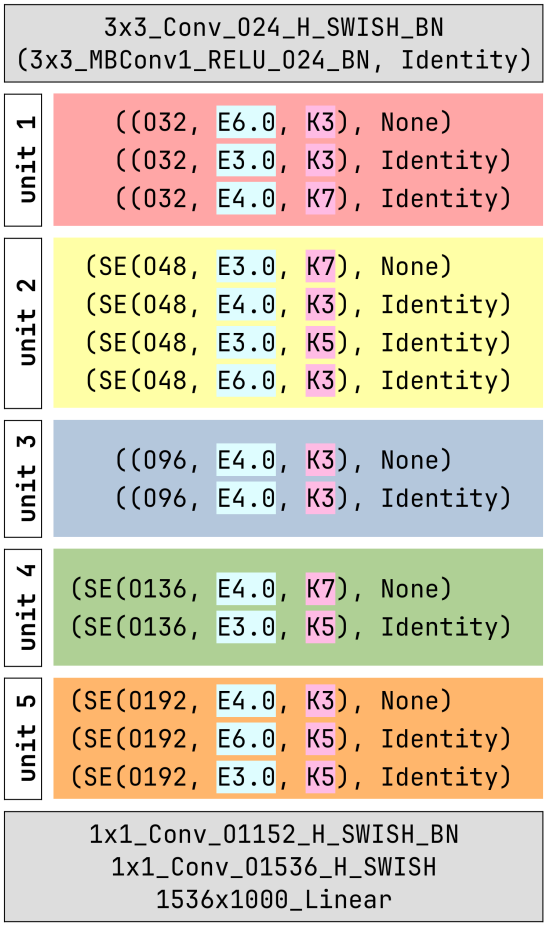


Figure 3.15 – Final architecture representation.

3.2.2 Objective Functions

The objective functions we want to optimize is the top-1 accuracy (maximize) and another conflicting objective, represented by a resource constraint, such as latency or FLOPS (minimize). For this, we use the accuracy and latency predictors provided by the OFA framework. That is, given the encoding of an architecture we use the accuracy predictor (the same regardless the hardware) to estimate the model accuracy, and we use the specific latency lookup table, for each hardware, to get the predicted latency.

Accuracy Predictor

The accuracy predictor is a simple feedforward neural network with three layers and 400 hidden units in each layer. The input of the accuracy predictor is a neural network encoded into a one-hot vector based on its kernel size and expand ratio. This accuracy predictor can speed up the evolutionary search during the searching stage, since getting the output of a three layer network is computationally much cheaper than really evaluating the architecture in all images of the validation set of the ImageNet dataset.

Efficiency Predictor

A latency predictor was also built to speed up the network search. However, instead of using a neural network to predict the latency, a latency lookup table is used (Cai et al., 2019). A FLOPS counter function can be used if we want the FLOPS as one of the objective functions related to resource constraints. There are a few hardware options available to choose the latency lookup table, such as the Google Pixel 2 mobile phone, the NVIDIA GTX 1080 Ti GPU, CPU or even FLOPS. We chose the Samsung Galaxy Note 10 for our experiments, considering the latency as the metric to be optimized alongside the accuracy.

3.2.3 Operators

In this section, we define the four operators that will take place on the individuals during the iterations of evolutionary algorithms: sampling, mutation, crossover and selection. The first three operators are the same for the three multi-objective optimization algorithms used to find the final architectures of the OFA search stage (namely NSGA-II, SMS-EMOA and SPEA2, discussed in the next section).

Sampling

The sampling operator is related to the initialization of the algorithm, that is, it defines the initial population of individuals at iteration zero. In our case we simply used the

random sampling, meaning that values for the depth, width and kernel size were randomly chosen among their respective valid choices (`d` $\in \{2, 3, 4\}$, `w` $\in \{3, 4, 6\}$, `ks` $\in \{3, 5, 7\}$).

Mutation

The mutation operator is responsible for implementing a local search, randomly perturbing selected individuals, and thus promoting diversity among the solutions, which might prevent the algorithm from getting stuck in a local minimum. In our experiments we defined that each gene of the chromosome has a probability of 10% of replacing its value into one of the valid choices (including the same value), which is the same probability used by the evolutionary search proposed on the OFA framework. Figure 3.16 illustrates the random mutation operator.

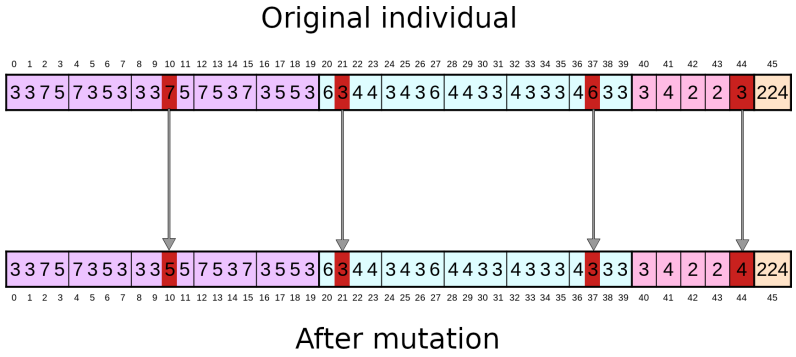


Figure 3.16 – Random mutation operator.

Crossover

The crossover operator, also known as recombination, takes two solutions as parents and combine them to generate a child solution. Here we chose the uniform crossover, which means that the value of each gene of the child solution is randomly taken from one of the parent solutions with equal probability. Figure 3.17 illustrates the uniform crossover (recombination) operator.

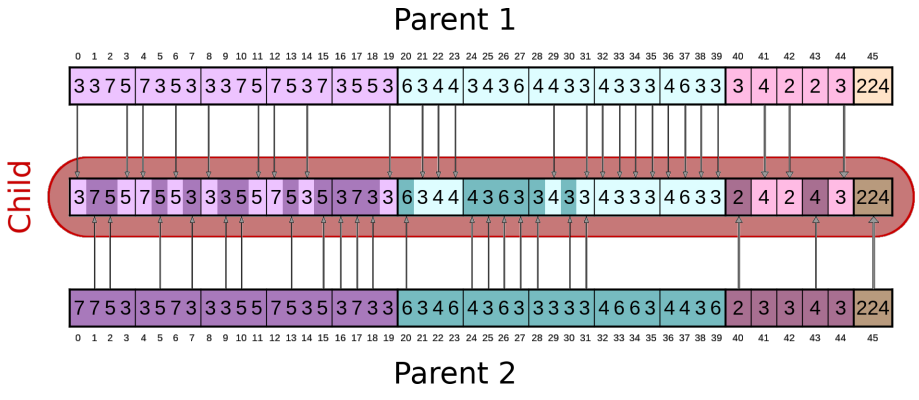


Figure 3.17 – Uniform crossover (recombination) operator.

Selection

Finally, the selection operator defines a criterion for choosing the individuals of the current population that will be used to generate the offspring, that is, the next generation of individuals. It usually incorporates the fitness function, taking the multiple objectives as guiding information, and in our case this operator is implicitly defined according to which of the three MOO algorithms is used during the evolutionary search.

3.2.4 Evolutionary Multi-Objective Algorithms (EMOA)

We are going to briefly describe the three EMOA considered here as candidates for the search engine. Evidently, any other EMOA could have been chosen instead.

NSGA-II

The first MOO algorithm considered is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002). In this algorithm the selection operator chooses first the individuals based on their level of non-dominated front. When the sum of the individuals already selected with the amount of individuals of the current front being considered surpass the population size previously defined, then the individuals of this current front are selected based on their crowding distance (Manhattan distance in the objective space). The selection based on the rank of the front always favors the non-dominated solutions with respect to the objective functions, while the crowding distance selection aims to spread the solutions toward regions less explored. In our experiments, we defined the population size to be 100 and ran the algorithm for 10,000 generations.
Some works already proposed the NSGA-II algorithm in the context of NAS. NSGA-Net (Lu, Whalen, et al., 2020), NSGANetV1 (Lu et al., 2021) and NSGANetV2 (Lu, Deb, et al., 2020) are some examples. However, they all have a different search space when compared to the one from this work, which uses the already trained OFA supernetwork as the basis for the searching stage. The NAG (Guo et al., 2021) shares the same search space of the OFA network, but their goal is to produce an architecture generator capable of sampling efficient networks given a specific budget. Furthermore, none of these works considered the scenario of decoupling the search and the deployment stages, proposed in this work.

SMS-EMOA

The second MOO algorithm considered is the SMS-EMOA (Beume et al., 2007). This algorithm guides the search aiming to maximize the hypervolume measure (Guerreiro et al., 2021), also known as s-metric, which is commonly applied to compare the performance of different evolutionary multi-objective optimization algorithms (EMOA). This algorithm

combines both the concepts of non-dominated sorting and the hypervolume measure as the selection operator. Similarly to the previous scenario, we used a population size of 100 and ran this algorithm for 10,000 generations.

SPEA2

Finally, the last EMOA considered is the SPEA2 (Zitzler et al., 2001), which is an improved version of the SPEA (Strength Pareto Evolutionary Algorithm) algorithm (Zitzler & Thiele, 1999). Here, the favoring of non-dominated solutions and their scattering along the Pareto frontier follow criteria also based on non-dominance and on the density of solutions in the objective space. Again we use the same hyperparameters: 100 individuals and 10,000 generations.

3.3 Ensemble formulation

The search stage is performed considering three different search strategies for comparison and validation purposes, as it will be explained in details in Chapter 4. Thus, after the search stage we end up with three final populations of neural networks, one for each search method. We then designed a series of experiments grouping different individuals from these populations to form ensembles (Hansen & Salamon, 1990). The idea is to find a set of neural networks whose individuals complement each other (Perrone & Cooper, 1995), reducing the variance of the error at the output, thus leading to better machine learning models (Geman et al., 1992).

3.3.1 Populations

There are three scenarios considered for the experiments, regarding the group of neural architectures in which the components of the ensemble will be taken from: the population from the random search, the population from the OFA search and the population from the multi-objective optimization framework OFA^2. Our hypothesis is that selecting efficient architectures to form the ensemble can perform better than ensembles with random or dominated individuals.

Random components

This group of architectures represents 100 neural networks sampled from the OFA search space without any search procedure. We randomly choose the value of each gene (depth, kernel size, expansion ratio) among its valid options. The procedure to get these architectures is the same of the one to define the initial population in the MOO algorithms.

OFA components

The second group of architectures is the one obtained from the evolutionary algorithm of the original OFA framework. Since this search is guided by a specific hardware constraint, we need to perform a full search for each requested architecture, resulting in 9 runs of the search algorithm for the 9 architectures in this group. The restrictions used as input of the searches are the latencies from 15 ms up to 55 ms, increasing 5 ms at each step.

OFA² components

The last set of architectures considered in the ensemble experiments are the architectures obtained from a Once-for-All multi-objective search performed over the supernetwork of the OFA framework. This is the motivation for the designation of OFA², because the search stage is also an OFA stage implemented by an EMOA, which aims to optimize both accuracy and latency. Since the evolutionary algorithm is population-based, all architectures are found at once at the end of the search procedure, being an approximation of the Pareto frontier. Each of them represents distinct trade-offs of the conflicting objectives. For the experiments with ensembles, we considered only the NSGA-II for the OFA² search.

3.3.2 Voting schemes

In order to evaluate the performance of the ensemble, we use two voting schemes: the hard voting and the soft voting, which are explained as follows.

Hard voting

In the first voting strategy, called "hard voting", the output of the ensemble is decided according to the most-voted top-1 class among the participants of the ensemble. If there is a draw in votes between one or more classes, we then check the occurrences of these classes on the second most likely output of each model (top-2 output) and decided by the most frequent. If there is still a draw in votes, then we keep checking the top-3 up to the top-5 output. After that, if the draw still persists, we choose the top-1 output provided by the largest subnetwork (according to the premise that larger models have more flexibility and might be more reliable than smaller models). In this scheme, the output of each neural network has the same importance, regardless how certain the model is about its output. Figure 3.18 illustrates the hard voting mechanism.

Soft voting

The other voting scheme also takes into account the probability assigned to each class on the output. For this, we take the last layer of each neural network and append a softmax

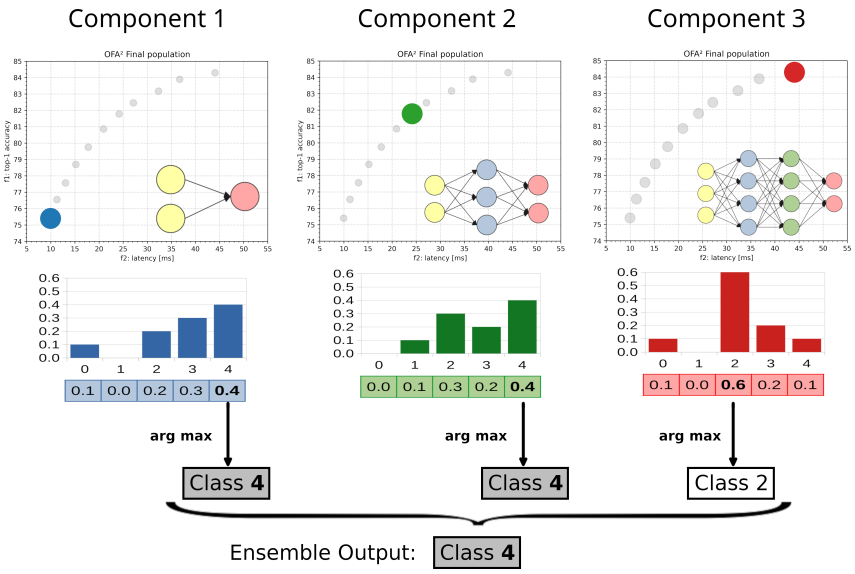


Figure 3.18 – Ensemble output with hard majority voting.

layer straight after it. To decide the output of the ensemble, we sum the probabilities for each class among all participants of the ensemble, and take the output with highest accumulated value. This helps to alleviate the problem of the hard voting scheme, which is the fact that a vote from a model with a low confidence in its output has the same weight of a vote from a model with a high confidence. This voting scheme, called "soft voting", provides a way to weight the vote of each architecture according to the confidence of the model in its output class, which can be beneficial in some cases. Figure 3.19 illustrates the soft voting mechanism.

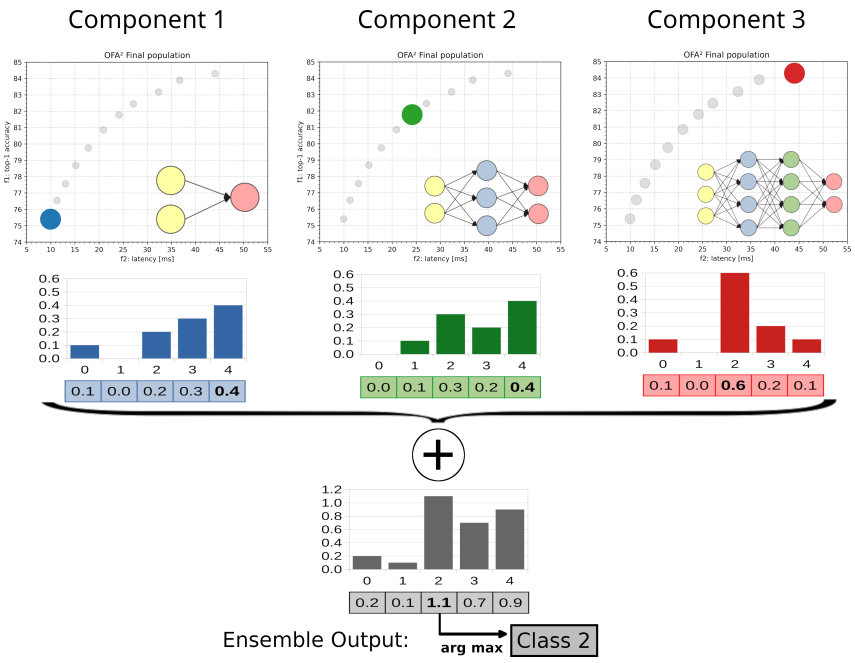


Figure 3.19 – Ensemble output with soft majority voting.

3.3.3 Latency

Concerning the latency of the ensemble, we consider two different approaches: the summed latency and the maximum latency, which are explained as follows.

Summed latency

In the first approach, the latency of the ensemble is defined to be the sum of the individual latencies of all architectures participating in the ensemble. This strategy is based on the worst-case premise that we have a single hardware with limited amount of memory to implement the ensemble, and therefore we need to load each model one at a time to evaluate its performance. Figure 3.20 illustrates the summed latency scenario for ensembles.

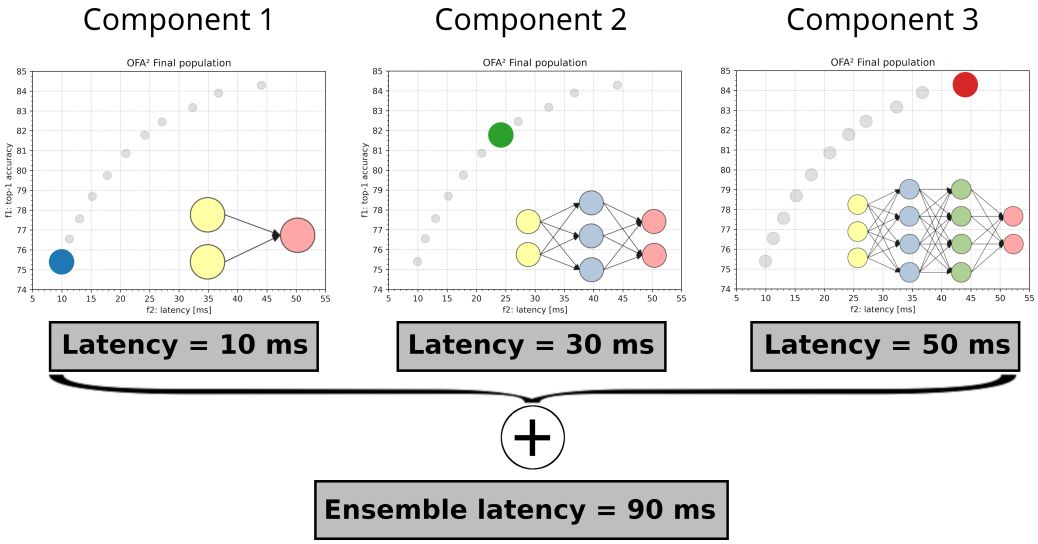


Figure 3.20 – Scenario considering the summed latency.

Maximum latency

In the second approach, the latency of the ensemble is equal to the model's latency that has the highest value among the networks which are members of the ensemble. This strategy is based on the premise that parallelization is viable, and therefore all models can be evaluated simultaneously. This, of course, requires a limited amount of models in the ensemble, due to memory scaling. As a consequence, we test ensembles with the number of components varying from 2 up to 8 (totalizing 7 different ensemble sizes) due to this imposed limitation. Figure 3.21 illustrates the maximum latency scenario for ensembles.

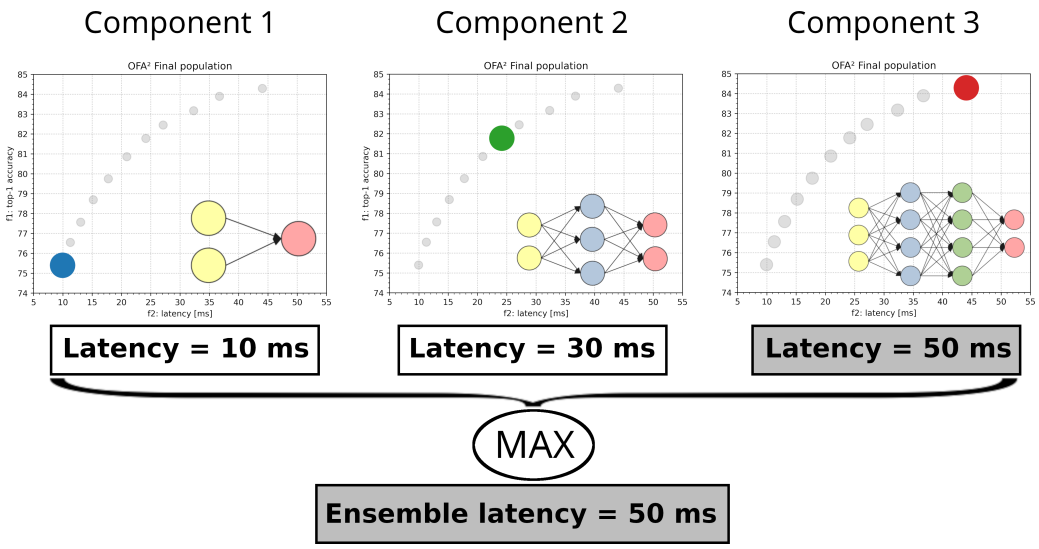


Figure 3.21 – Scenario considering the maximum latency.

3.3.4 OFA³: Genetic algorithm to select the components of the ensemble

The last set of experiments are still related to ensembles, but instead of randomly or deterministically sampling neural networks from a population to form the ensemble, we propose to solve the problem of automatically determining how many and which available non-dominated subnetworks are going to compose the ensemble, by implementing another multi-objective search which we called OFA³.

Algorithm

Following a similar approach to the OFA² search, we decided to work with the NSGA-II, SMS-EMOA and SPEA2 algorithms here. Clearly, other MOO algorithms could also be used at this step. The idea of using an evolutionary algorithm to obtain ensembles is not innovative (Liu et al., 2000). However, most of the approaches use the evolutionary process just to create diversity among the individuals of the ensemble and does not propose the ensemble formation problem as a multi-objective problem itself (Brown et al., 2005) (Brown, 2004).

Dataset

The dataset used for the OFA³ search is the same as for the OFA and OFA², being the ImageNet dataset. However, since the validation set of the ImageNet containing 50,000 images will be used to evaluate the ensembles and compare them against the single architectures, a new subset of images was necessary to perform the OFA³ search. For this, we simply took the first 50 images of each one of the 1,000 classes of the training set of the ImageNet and group them to form the 50,000 images that will be used to guide the search for the ensembles.

Objective functions

The objective functions optimized are the same as before, accuracy and latency, considering both the sum and maximum latency approaches, but just the soft voting scheme to decide the output of the ensemble. Since the OFA project provides a FLOPS counter for the architectures, it is also possible to optimize the accuracy and FLOPS jointly.

Encoding

The set of candidate individuals to form the ensemble is the population obtained from the OFA² search, totalizing 100 efficient neural networks with different trade-offs between accuracy and latency. The encoding used is simply an array with 100 binary genes, one for each neural network from the pool of 100 candidate architectures to form the ensemble, meaning that if a specific gene has a value 0, then the neural network represented by that gene will not compose the ensemble, and if the gene has a value 1, then the neural network at that index compose the ensemble. Figure 3.22 illustrates three examples of encodings and their respective sets of neural networks composing the ensembles. The encoding in blue has a single gene with the value 1 in its first position, meaning that only the first neural network of the population will compose that ensemble. The encoding in green has three consecutive genes with the value 1 in the middle of the array, meaning that the three neural networks represented by these genes will take part in the ensemble. The encoding in red has all genes at value 0, except the last one, meaning that only the last neural network of the population composes this ensemble. The algorithm ran for a total of 2,000 generations and the results are discussed in the next section.

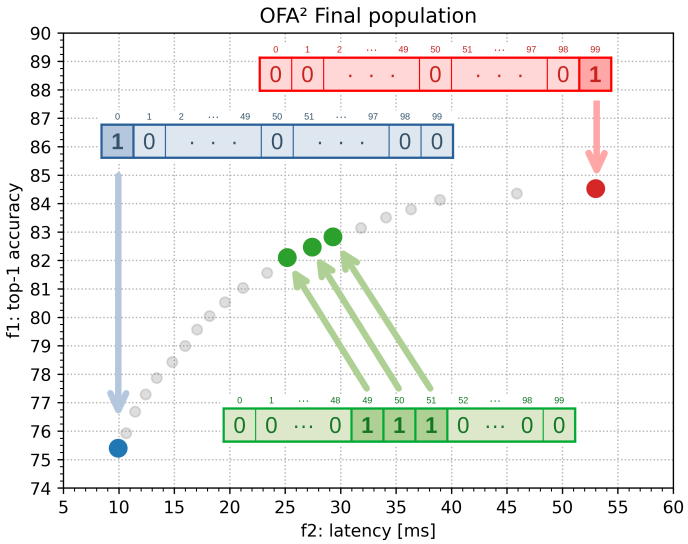


Figure 3.22 – Encoding for the multi-objective search responsible for automatically defining how many and which non-dominated subnetworks will compose the ensemble.

4 Experiments and Obtained Results

The experiments presented in this section are divided in two parts. In the first part, the focus is on the search stage of the OFA framework and how to improve it in order to get more efficient architectures in terms of both accuracy and latency. Then, in the second part, the experiments are related to the formation of ensembles, aiming at finding a good strategy to select specific neural networks to be part of the ensemble, providing a better accuracy and latency, on the best scenarios, or improving the accuracy at the cost of increasing the latency.

4.1 Search strategies

We consider three search strategies to explore the OFA search space. The first one consists of taking a population of random neural networks, the second one is the original evolutionary search of the OFA framework, and finally, the last strategy consists of our proposed multi-objective optimization OFA² search method. Next, we will describe each of those strategies and show their performance, followed by some comparative analysis.

4.1.1 Random search

For the random strategy, the first step is to sample random architectures from the OFA search space and calculate the accuracy prediction for each of them. Then, in order to compare the accuracy predictor with the real performance of the neural network, we take each of these architectures and evaluate them on the ImageNet validation set. After plotting the predicted and evaluated accuracies for each of these neural networks, we found out that the predicted accuracies have an offset over the evaluated accuracies, although the overall format of the points' distribution are similar. This offset can be explained due to an overfitting caused on the accuracy predictor, since it was trained in a subset of 10k images of the same training set used to train the OFA super-network weights. In fact, to verify that this gap can be approximated to a constant, we simply took the predicted accuracies and subtracted an offset of 4.5% in the top-1 metric. Figure 4.1 shows the predicted accuracy in light blue, the predicted accuracy with the offset of 4.5% subtracted in dark blue, and the real evaluated accuracy in orange, for the random population. This offset characteristic will also appear with the original OFA search and in our search method OFA², as will be shown later. Fortunately, our evolutionary approaches are not sensitive to this detected offset, given that the evolutionary paradigm is founded on the relative fitness of individuals.

Figure 4.1 – Predicted and evaluated accuracies of the random subnetworks.

4.1.2 OFA search

For the OFA population, we used the original evolutionary strategy of OFA framework (Cai et al., 2020). The operators used by this strategy is random mutation with 0.1 probability for each hyperparameter, and a uniform crossover. The population size is 100 individuals and the search process runs for 500 generations. Figure 4.2 shows the performance of the architectures found by the algorithm. The dots in light blue represents the accuracy prediction of each architecture, the dots in orange are the real accuracies evaluated and the dots in dark blue are the predicted values subtracted by the offset of 5.0%. It is important to note that one search procedure was done for each architecture found. That is, we had to perform an architecture search for the target latency of 15 ms, then repeat the process for the target latency of 20 ms, and so on and so forth, up until the target latency of 55 ms. This leads to 9 runs of the OFA search algorithm in total, resulting in 9 architectures represented by the dots in the figure.

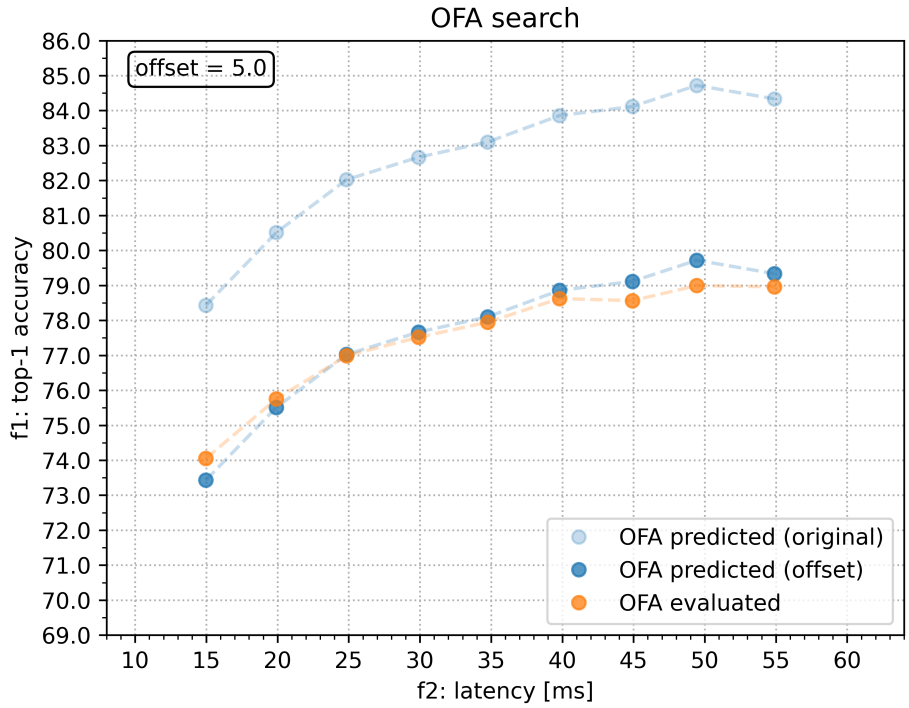


Figure 4.2 – Predicted and evaluated accuracies of architectures produced by each run (latency constraint varies along the runs) of the original OFA search.

4.1.3 OFA² search

For this proposed search method, we solved the multi-objective optimization problem formulated for the search stage of the Once-for-All framework using three alternative EMOA: NSGA-II, SMS-EMOA and SPEA2. Figures 4.3, 4.4 and 4.5 show the progression of the populations along the iterations for the NSGA-II, SMS-EMOA and SPEA2 algorithms, respectively. We can see that the individuals of the initial population (red) are spread across the objective space, which is comprehensible, since these individuals are randomly sampled from the OFA search space. Even though they are far from being efficient in terms of both accuracy and latency compared to the non-dominated final solutions, over the generations they progressively approximate the Pareto front, as we can see for the individuals after 10 (green), 100 (orange) and 10,000 generations (blue). This indicates that even though the neural networks of the OFA search space have their weights already trained, a search is indeed advantageous to retrieve efficient architectures. We ran this search method for three different random seeds. However, the results presented here are all related to one specific random seed, out of the three.

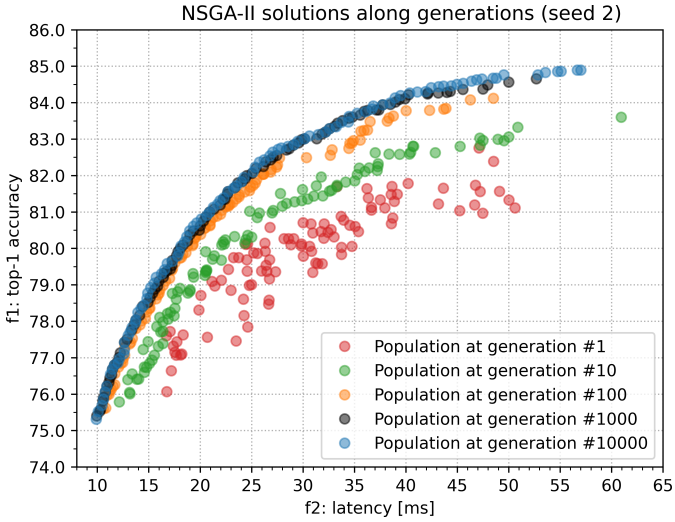


Figure 4.3 – Progression of the solutions for the NSGA-II algorithm.

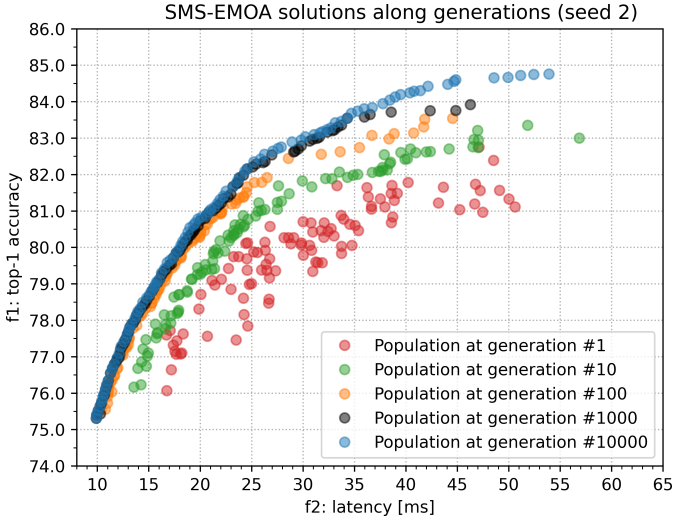


Figure 4.4 – Progression of the solutions for the SMS-EMOA algorithm.

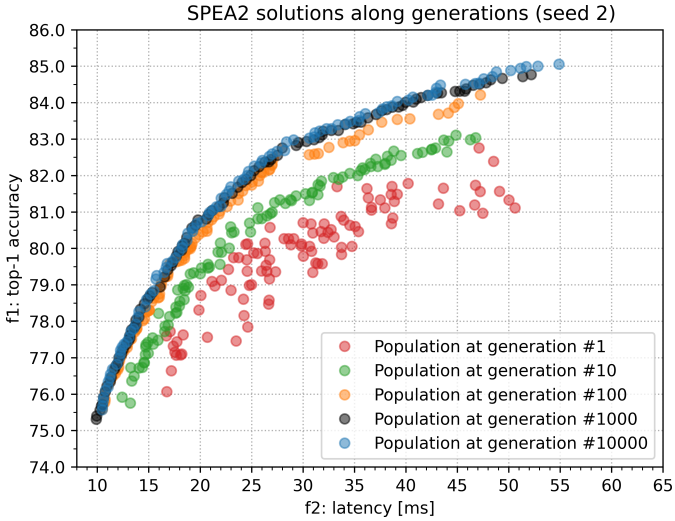


Figure 4.5 – Progression of the solutions for the SPEA2 algorithm.

Aiming at comparing these two evolutionary algorithms, Figure 4.6 shows the final architectures found by each of them after 10,000 generations. We can see that the final population approximates the typical Pareto front for MOO problems with two conflicting objective functions. Moreover, we can see that while the SMS-EMOA finds slightly better architectures with lower latency, the NSGA-II and SPEA2 find better architectures at the end of the latency axis. Apart from the region with higher latency, the predicted accuracies found by the three algorithms are very similar.

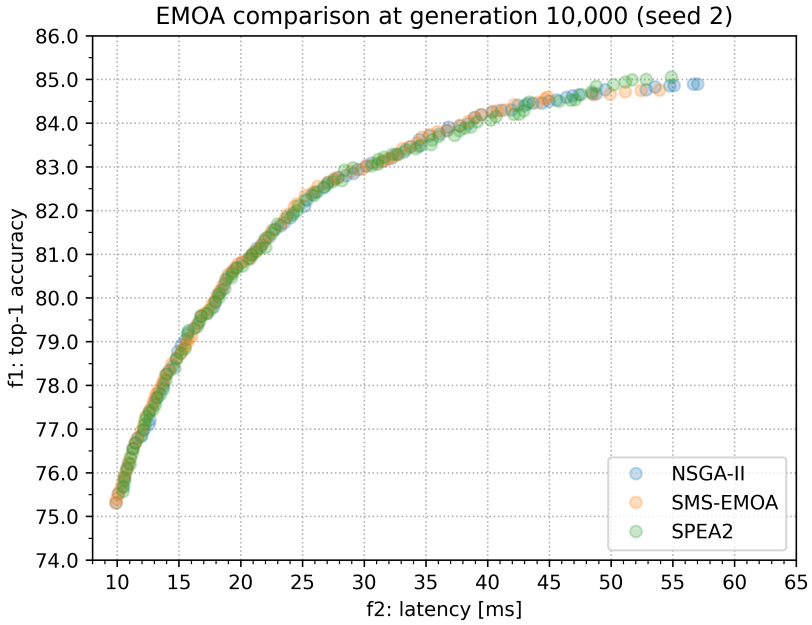


Figure 4.6 – Comparison between NSGA-II, SMS-EMOA and SPEA2 final populations.

In Figure 4.7 we plot the progression of the hypervolume measure (s-metric) of the NSGA-II, SMS-EMOA and the SPEA2 along 10,000 generations. This metric is commonly used to compare the performance of different evolutionary multi-objective optimization algorithms (EMOA) and requires a reference point on the objective space to be calculated, which is taken as the point (100, 25).

In an attempt to avoid a kind of underfitting and a kind of overfitting, we have NSGA-II as the algorithm with intermediary value for the hypervolume measure along generations and particularly at the end of 10,000 generations of the evolutionary search. Therefore, we have decided to use the population found by this algorithm for the following experiments with the ensembles. Figure 4.8 shows the accuracies of the architectures obtained from the OFA^2 search. The dots in light blue are the accuracies using the accuracy predictor, the dots in dark blue are the predicted accuracies minus the offset of 5.5%, and the dots in orange are the real evaluated accuracies.

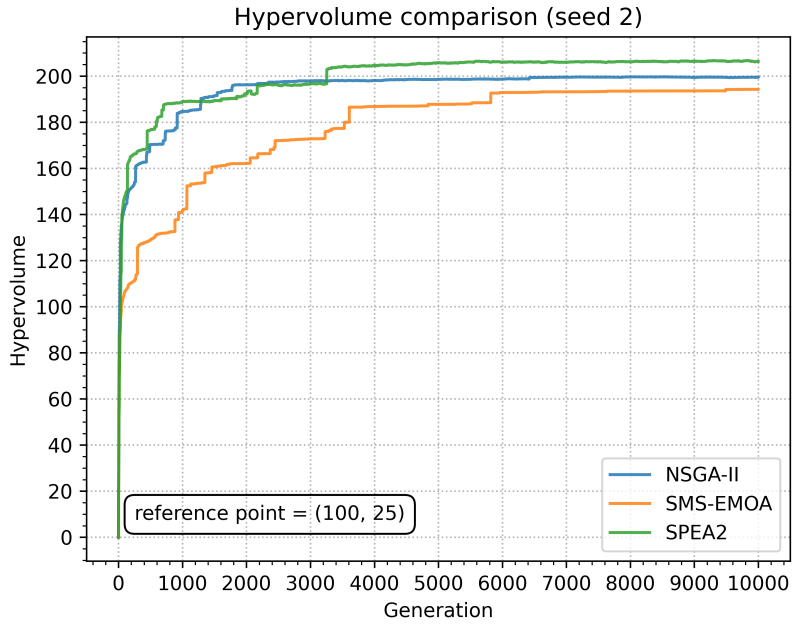


Figure 4.7 – Comparison between NSGA-II, SMS-EMOA and SPEA2 hypervolumes along generations.

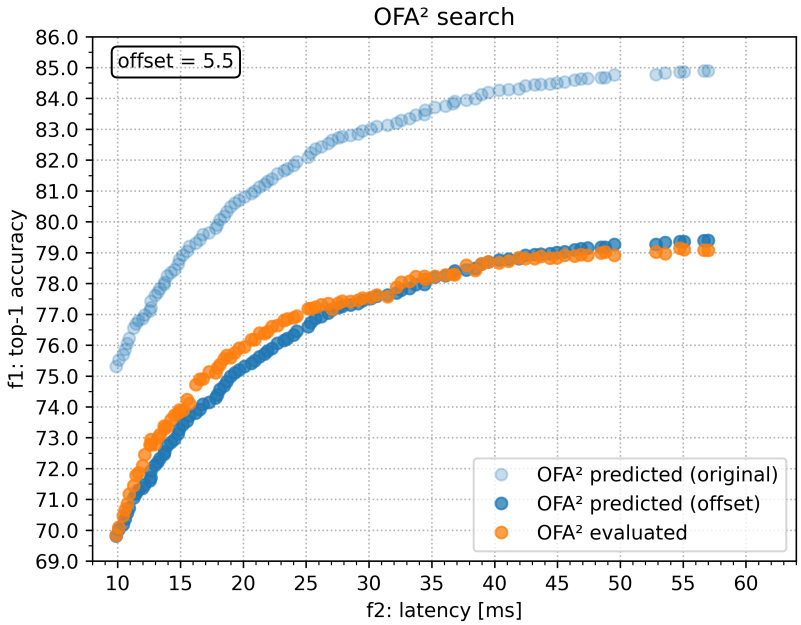


Figure 4.8 – Predicted and evaluated accuracies of architectures from OFA² search.

4.1.4 Comparative Analysis

Now that we have the final architectures of each search method, we can plot them together in a single graph. Figure 4.9 shows the comparison between the 100 individuals of the final population searched with the NSGA-II for the OFA² search, the 100 individuals randomly sampled from the OFA search space and the 9 individuals obtained with 9 runs of the original evolutionary algorithm of the OFA framework for 9 different latency constraint (15 ms to 55 ms, with a step of 5 ms). We can see that the random sampled architectures perform reasonably well, with predicted accuracies all above 76% top-1, meaning that after the training procedure of the OFA framework, there is no need for retraining or fine-tuning the architectures. However, when we compare the results of the random search against those obtained from OFA and OFA² search methods, it is clear that an additional search procedure is required to obtain more efficient architectures.

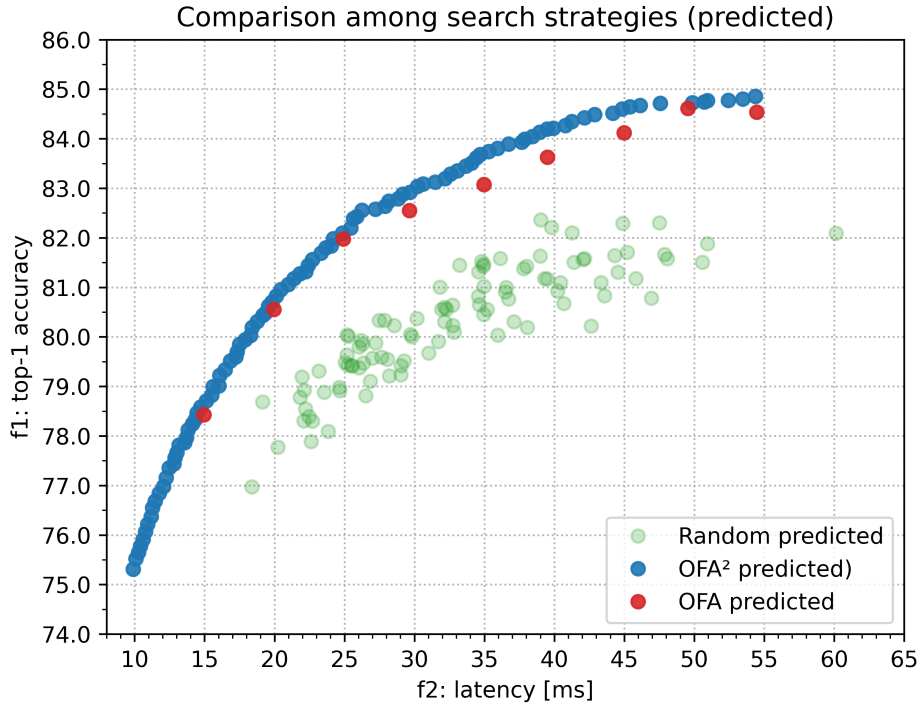


Figure 4.9 – Comparison of the search methods using the accuracy predictor.

The performance predictors are useful to speed up the search process. However, if the predictions are not accurate, the resulting architectures of the search process will not present the same performance during inference. In order to check the reliability of the accuracy predictor, we took each of the architectures shown in Figure 4.9 and measured their real accuracy under the validation set of the ImageNet. The results are shown in Figure 4.10. We can see that, although the top-1% accuracies are not exactly the same, the shape of the curves are similar, which means that the accuracy predictor works well disregarding a constant offset. Luckily, the domination concept used by the algorithms

tested are not dependent on the absolute values of the objective functions, relying instead on relative values, which means that as long the shape of the curves on the objective space is the same, we should not have any problem with these types of EMOA.

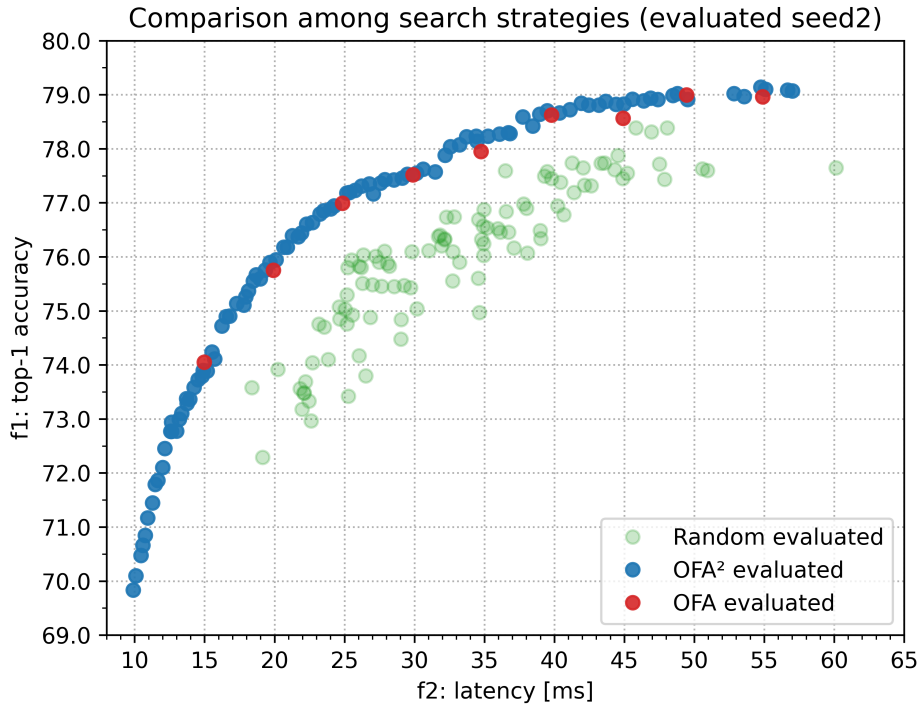


Figure 4.10 – Comparison of the search methods for the real evaluated accuracy.

Table 4.1 – Comparison of ImageNet results between OFA and OFA² hardware-aware NAS search methods.

	ImageNet Top-1% under latency constraints										Search cost
method	10 ms	15 ms	20 ms	25 ms	30 ms	35 ms	40 ms	45 ms	50 ms	55 ms	GPU hour
OFA	N/A	**74.05**	75.75	**76.99**	**77.52**	77.95	78.62	78.62	79.00	79.00	0.83
OFA²	**69.83**	73.79	**75.76**	76.89	77.46	**78.23**	**78.64**	**78.88**	**79.02**	**79.02**	**0.01**

Table 4.1 presents the results for the OFA and OFA² searches. We can see that for most of the latency constraints, OFA² performs better than OFA. The computational costs of all methods are negligible when compared with the cost of training the Once-for-All super-network (1,200 GPU hours), but notice that the cost of OFA² is even more reduced, when compared with the cost of OFA. This expressive reduction is motivated by the fact that OFA² performs a single (Once-for-All) search and OFA requires an independent search per candidate solution.

4.2 Comparison of distinct ensemble compositions

4.2.1 Random components

The first group of architectures considered in our experiments with ensembles are those formed by the population of random architectures. We sample 43 random different combinations of these architectures to make up the committees. We start with ensembles containing 2 architectures, then we sample 43 random different combinations for ensembles with size 3, so on and so forth, up to 43 ensembles composed of 8 neural networks. This is done once for the hard voting scheme and once for the soft voting scheme, both considering the sum and maximum of latencies in the ensemble. Figures 4.11 and 4.12 illustrate the non-dominated ensembles with different numbers of neural networks for the summed latency approach, considering the hard and soft voting schemes, respectively, and Figures 4.13 and 4.14 illustrates the non-dominated ensembles for the maximum latency approach, considering the hard and soft voting schemes, respectively.

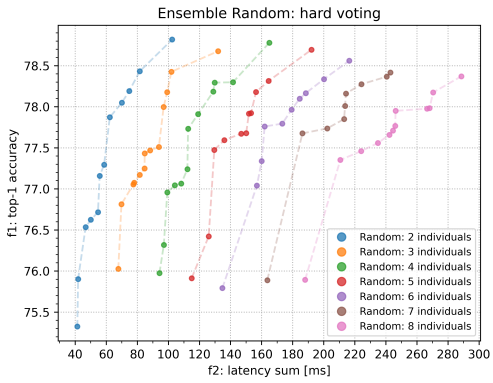
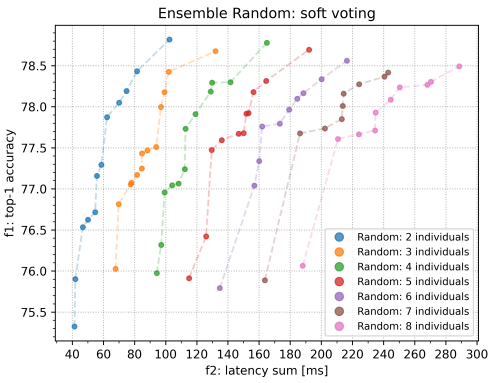
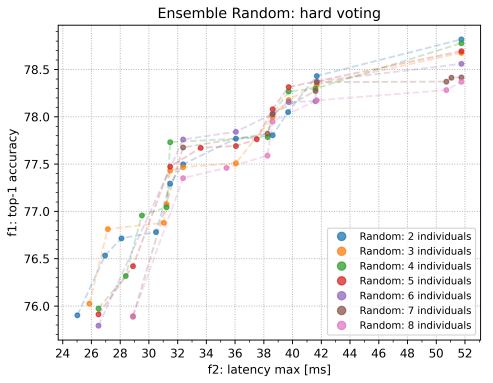
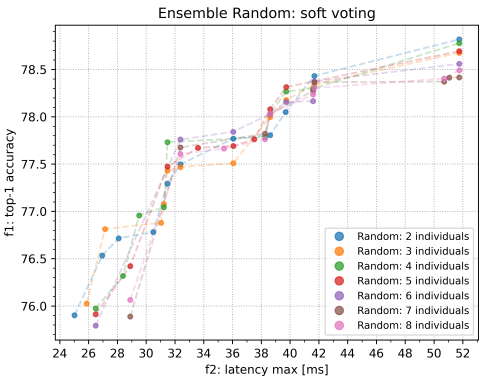
	hard voting	soft voting
latency sum		
latency max		

Figure 4.11 – Non-dominated random ensembles (latency sum, hard voting).

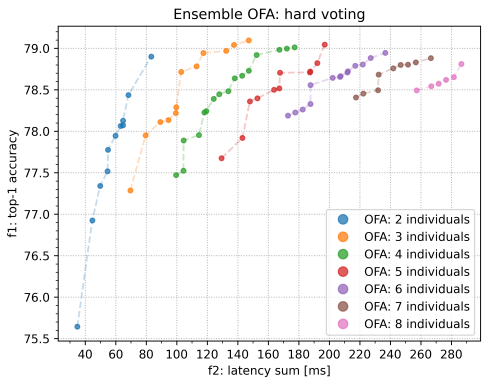
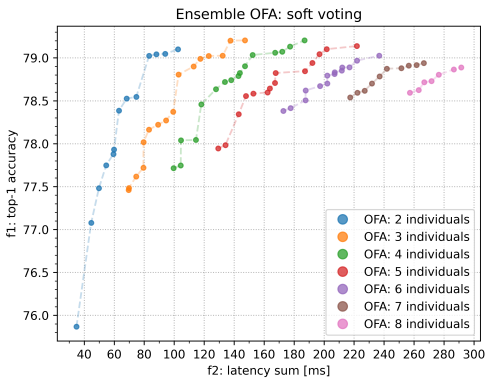
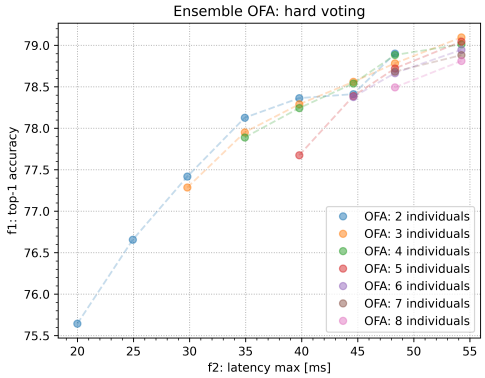
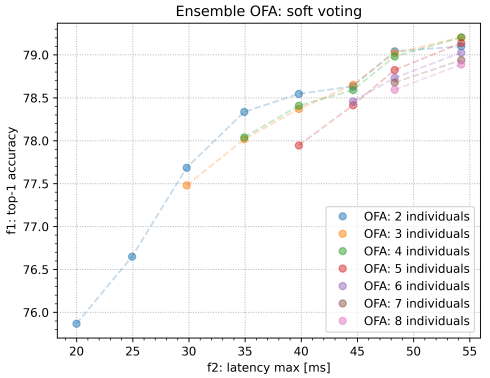
Figure 4.12 – Non-dominated random ensembles (latency sum, soft voting).

Figure 4.13 – Non-dominated random ensembles (latency max, hard voting).

Figure 4.14 – Non-dominated random ensembles (latency max, soft voting).

4.2.2 OFA components

The experiments with the ensembles formed by the architectures found by the OFA search are similar to the ones done with the random ensembles, with the difference that the OFA population has 9 neural networks, instead of the 100 candidates of the random population. We first sample 43 different ensemble combinations with 2 architectures considering both the hard and soft voting schemes and for the sum and maximum latencies approach. Then we sample 43 different ensembles with 3 neural networks, and keep doing it up to ensembles formed by 8 neural networks. Figures 4.15 and 4.16 illustrate the non-dominated ensembles with different numbers of architectures for the summed latency approach, considering the hard and soft voting schemes, respectively. Figures 4.17 and 4.18 illustrate the non-dominated ensembles for the maximum latency approach, considering the hard and soft voting schemes, respectively.

	hard voting	soft voting
latency sum		
	Figure 4.15 – Non-dominated OFA ensembles (latency sum, hard voting).	Figure 4.16 – Non-dominated OFA ensembles (latency sum, soft voting).
latency max		
	Figure 4.17 – Non-dominated OFA ensembles (latency max, hard voting).	Figure 4.18 – Non-dominated OFA ensembles (latency max, soft voting).

4.2.3 OFA² components

Finally, the last group of individuals considered in the ensemble experiments are those of the last population obtained from the OFA² search, using the NSGA-II algorithm. We then follow similar approaches done with the random and OFA population, in order to compare the results later.

Random selection of OFA² components

For the further comparison of the OFA² ensembles with the random ensembles, we simply follow the same selection strategy of the random approach. That is, we take 43 different combinations of architectures for each of the ensembles with 2 up to 8 models in its composition out of the 100 models available. Figures 4.19 and 4.20 illustrate the non-dominated ensembles for the summed latency approach, and Figures 4.21 and 4.22 illustrate the non-dominated ensembles for the maximum latency approach, considering the hard and soft voting schemes, respectively.

OFA-like selection of OFA² components

For the comparison with the OFA ensembles, we use a subset of the OFA² population with latencies similar to the ones produced by the architectures found by the OFA search. The first step is to sort the neural networks of the OFA² population by latency, which means that the model with index 0 is the model with the lowest latency (most left model in Figure 4.10) and the model with index 99 is the one with the highest latency (most right model in Figure 4.10). Next, we choose the architectures immediately before the latency constraints of 15 ms, then 20 ms, up to 55 ms, which are the same latency constraints used to find the 9 architectures of the OFA population. Then, we follow the same procedure adopted to build the OFA ensembles. That is, we sample 43 different combinations of 2 neural networks among the total of 9 available, then we repeat for ensembles with 3 neural networks, and so on up to ensembles with a total of 8 neural networks. Figures 4.23 and 4.24 illustrate the non-dominated ensembles considering the summed latency approach, for the hard and soft voting schemes, respectively. Figures 4.25 and 4.26 illustrate the non-dominated ensembles for the maximum latency approach, considering the hard and soft voting schemes, respectively.

Before comparing the performance of the ensembles regarding the search method that led to the population of neural networks used to select the components of the ensembles, there are some insights we can take from the previous figures. From the ensembles with random selection, we can see that both for the hard and soft voting schemes, and both summed and maximum latency approach, when the number of components on the ensemble increases, the performance decreases, as shown in Figures 4.11 to 4.14. This

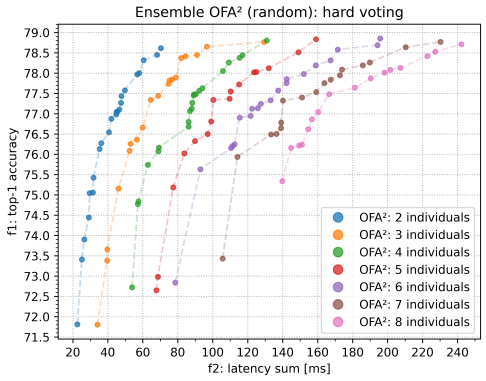
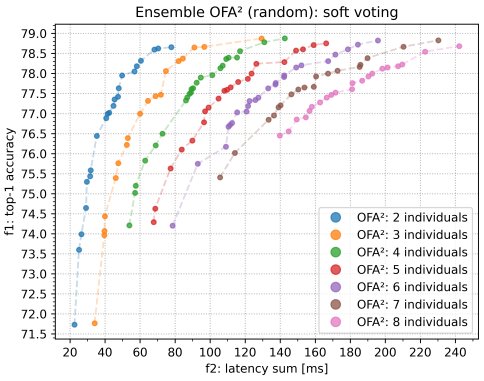
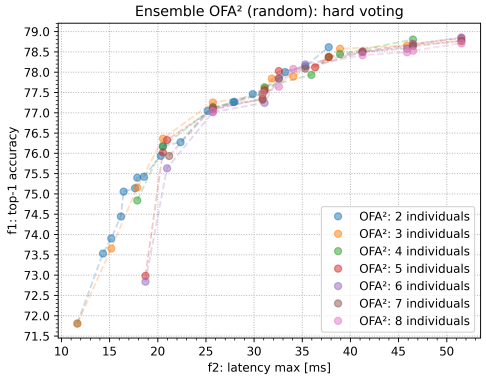
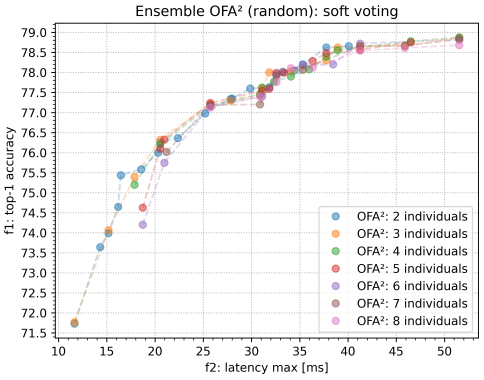
	hard voting	soft voting
latency sum		
latency max		

Figure 4.19 – Non-dominated OFA² ensembles (random, latency sum, hard voting).

Figure 4.20 – Non-dominated OFA² ensembles (random, latency sum, soft voting).

Figure 4.21 – Non-dominated OFA² ensembles (random, latency max, hard voting).

Figure 4.22 – Non-dominated OFA² ensembles (random, latency max, soft voting).

might sound counterintuitive at first, but it is already known that selecting a large number of components to form the ensemble may actually hurt the overall performance (Zhou et al., 2002). For the ensembles with the OFA population, we can see that the ensembles with 3 and 4 components are the ones with final better accuracy for both summed and maximum latency, and for low latencies requirements, the ensembles with 2 components are the winner, as illustrated in Figures 4.15 to 4.18. Again we can see that as the number of components present on the ensemble increases, the performance tends to degrade. This characteristic also happens with the OFA² with OFA-like components, but it is not so evident for the OFA² with random components, though, as illustrated in Figures 4.19 to 4.26, even though the best ensembles are still the ones with fewer components, from 3 to 6 in the case of OFA-like OFA² architectures.

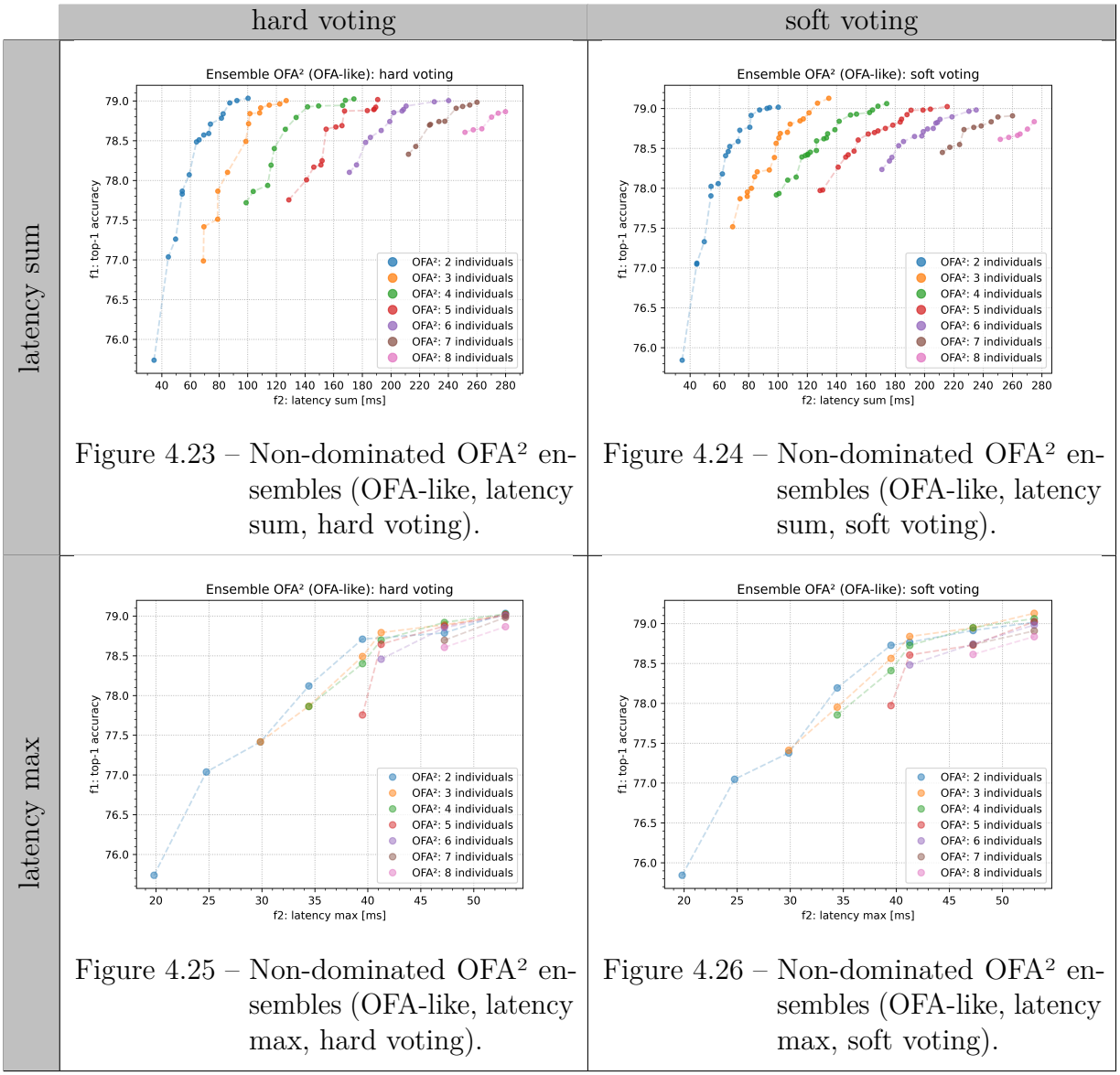


Figure 4.23 – Non-dominated OFA² ensembles (OFA-like, latency sum, hard voting).

Figure 4.24 – Non-dominated OFA² ensembles (OFA-like, latency sum, soft voting).

Figure 4.25 – Non-dominated OFA² ensembles (OFA-like, latency max, hard voting).

Figure 4.26 – Non-dominated OFA² ensembles (OFA-like, latency max, soft voting).

4.2.4 Comparative Analysis

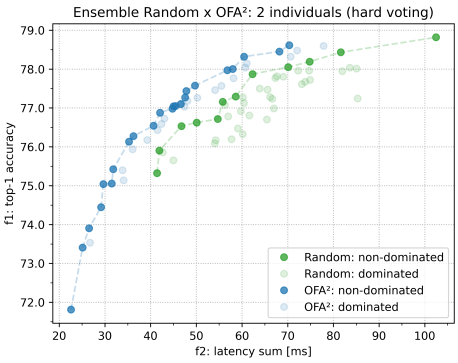
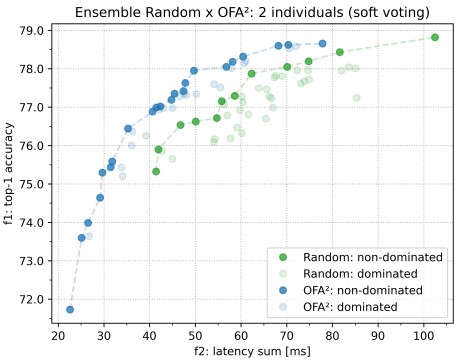
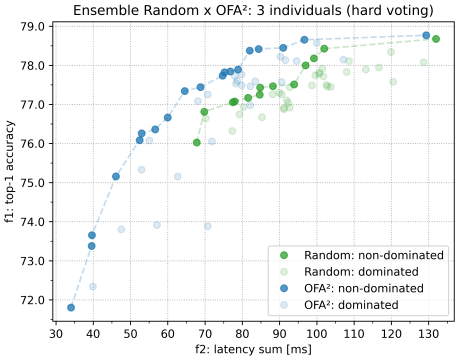
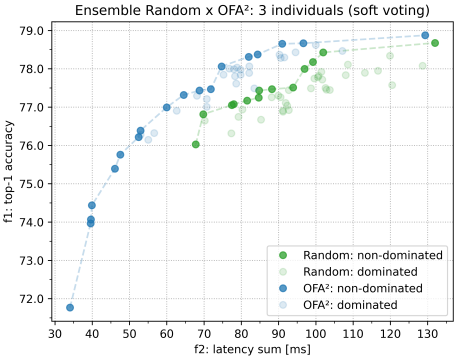
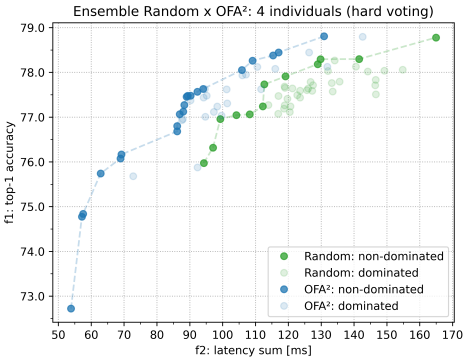
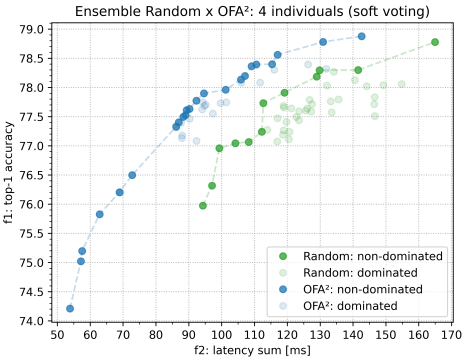
In what follows, we compare the ensembles regarding the search technique that guided to the population of neural networks candidates to be part of the committees. The comparisons are done with both hard and soft voting schemes and considering the summed and maximum latency approaches.

OFA² x Random

The first analysis compares the ensembles from OFA² and random populations. Table 4.2 shows the comparison for the summed latency approach and Table 4.3 shows the comparison for the maximum latency approach. The dominated ensembles are represented in light colors, while the non-dominated ensembles are represented in dark colors and connected through a dashed line. It is possible to affirm that the ensembles formed with OFA² neural networks, that is, the ensembles whose components are non-dominated neural networks,

consistently outperform the ensembles using the random search strategy, regardless the number of individuals on the ensemble, which was already expected (Kuncheva & Whitaker, 2003) (Kuncheva et al., 2003).

Table 4.2 – Ensembles with individuals from random and OFA² searches, considering the sum of the latencies of all individuals on the ensemble.

	Latency (sum)	
	Hard voting	Soft voting
2 individuals		
3 individuals		
4 individuals		

Continued on next page

Table 4.2: Ensembles with individuals from random and OFA^2 searches, considering the sum of the latencies of all individuals on the ensemble (Continued).

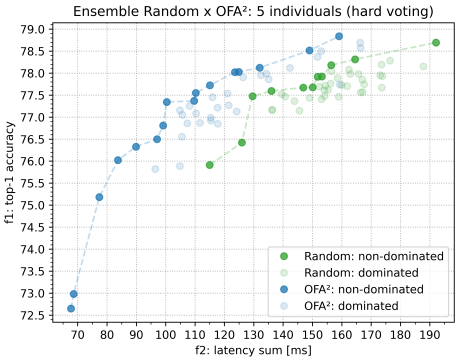
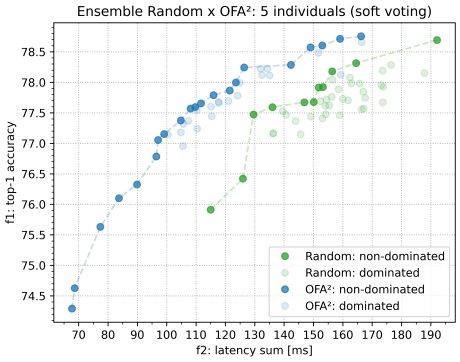
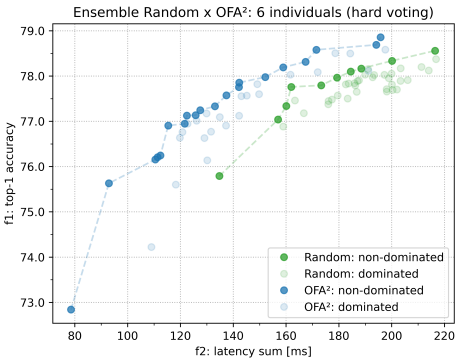
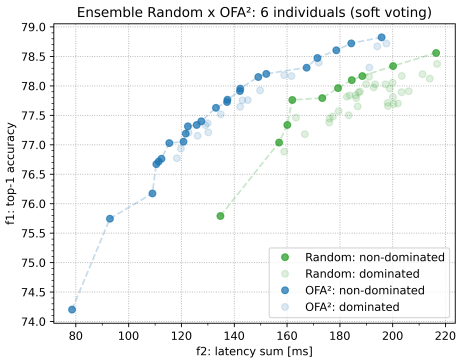
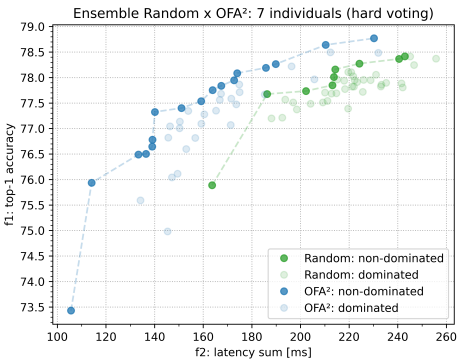
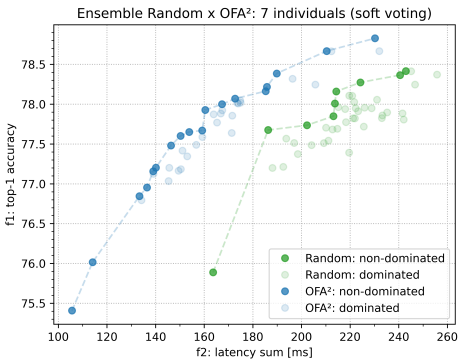
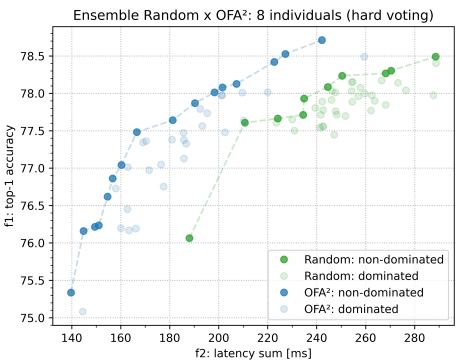
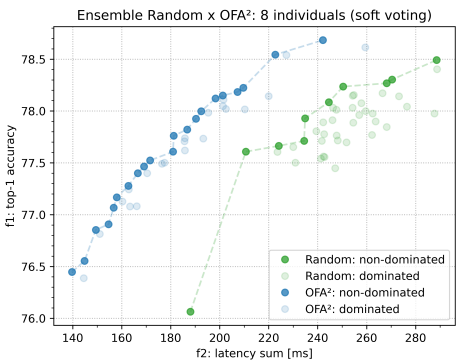
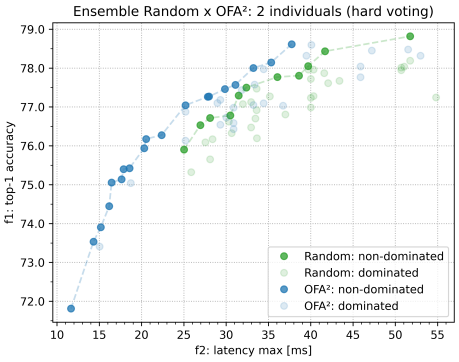
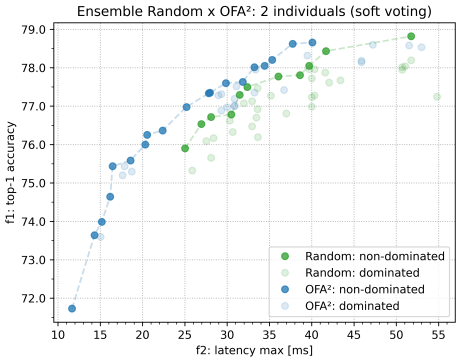
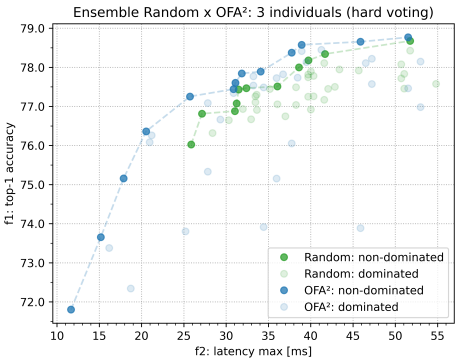
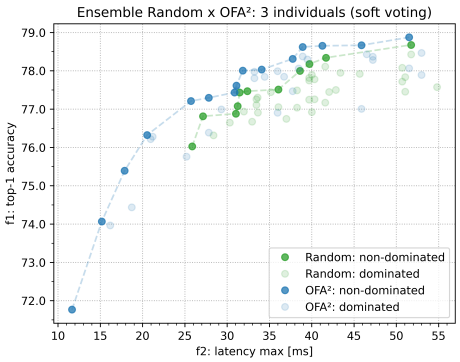
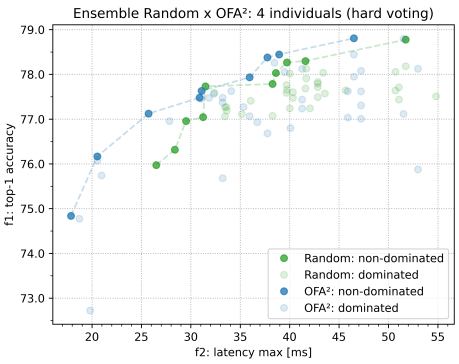
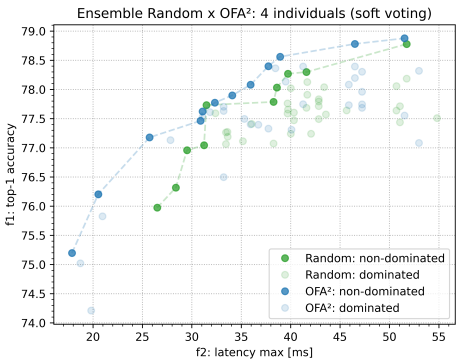
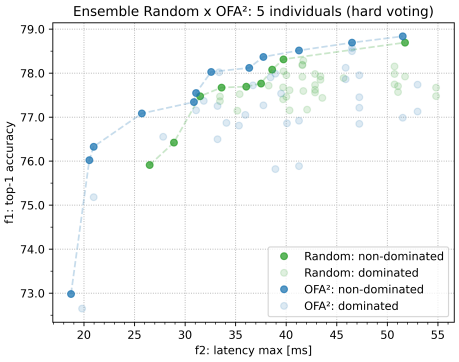
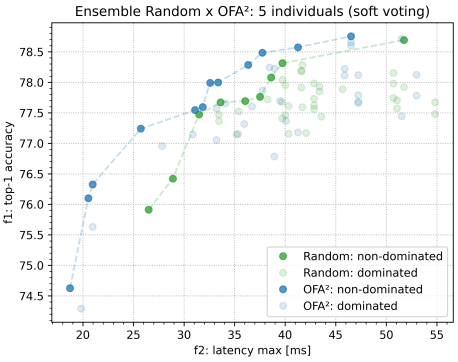
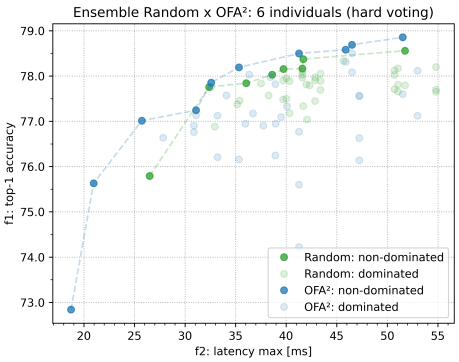
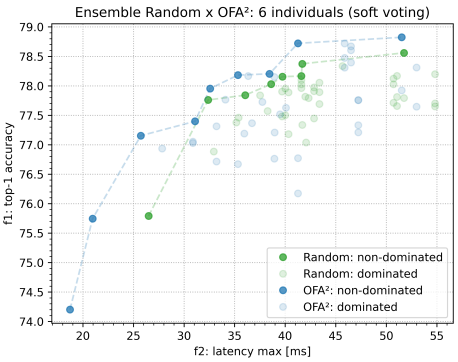
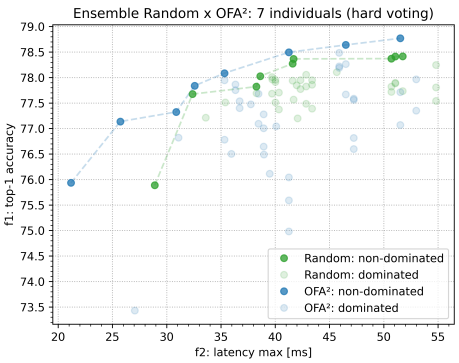
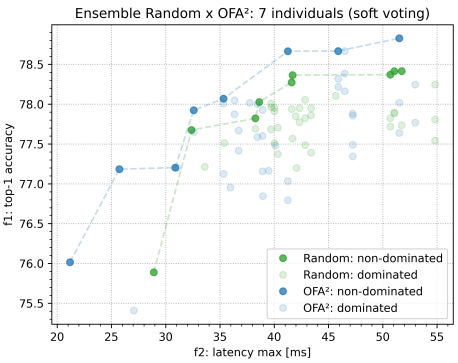
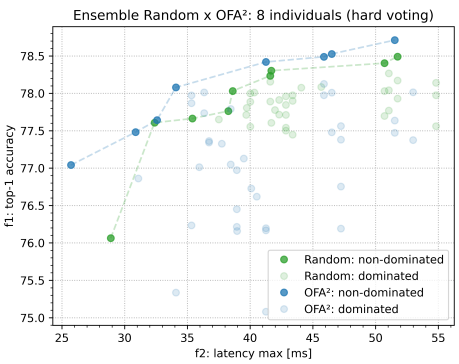
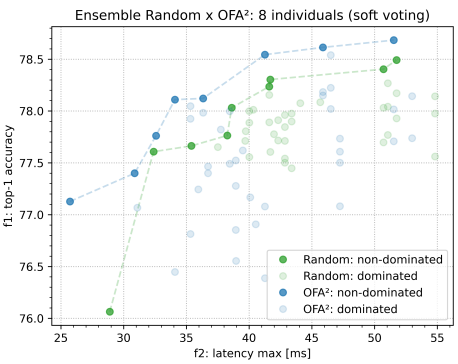
5 individuals		
6 individuals		
7 individuals		
8 individuals		

Table 4.3 – Ensembles with individuals from random and OFA^2 searches, considering the maximum latency of the individuals on the ensemble.

	Latency (max)	
	Hard voting	Soft voting
2 individuals		
3 individuals		
4 individuals		

Continued on next page

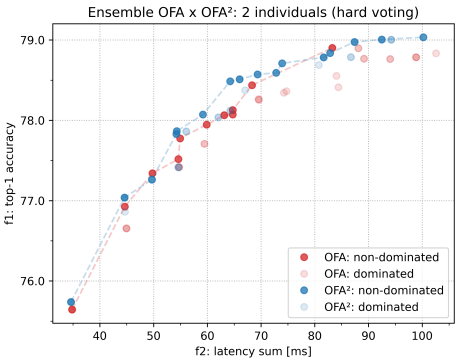
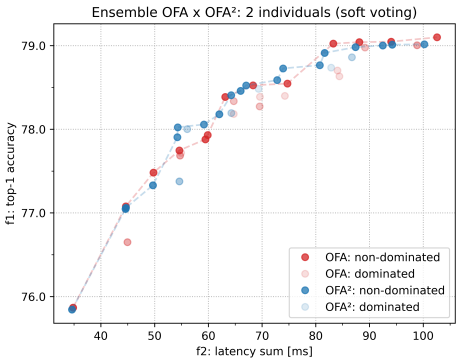
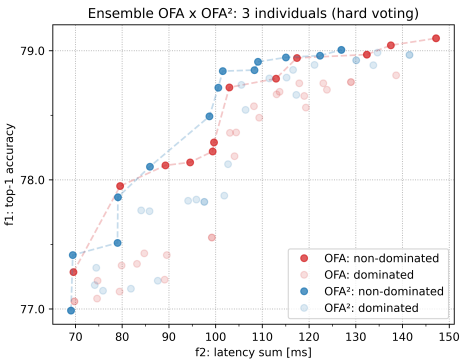
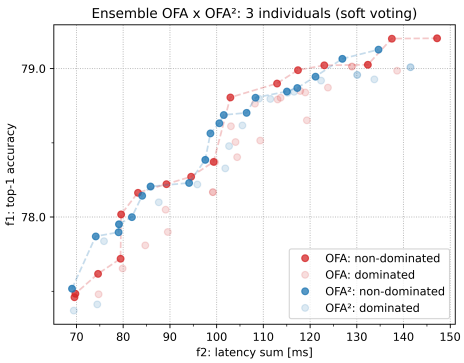
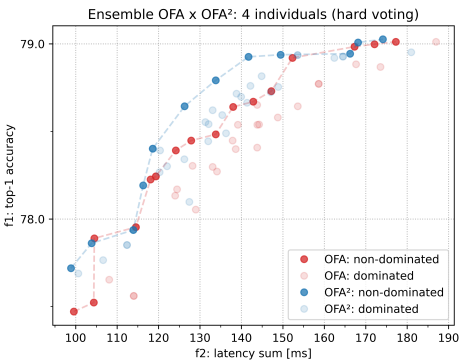
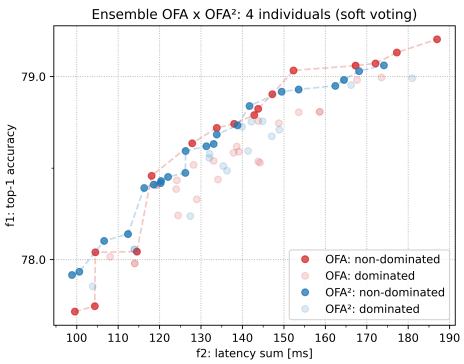
Table 4.3: Ensembles with individuals from random and OFA^2 searches, considering the maximum latency of the individuals on the ensemble (Continued).

5 individuals		
6 individuals		
7 individuals		
8 individuals		

OFA² x OFA

The second round of comparisons are between the OFA and OFA² searches. Table 4.4 shows the comparisons for the summed latency approach and Table 4.5 shows the comparison for the maximum latency approach. From the figures of both these tables we can see that for the hard voting scheme, the OFA² ensembles performs better that the OFA ensembles in almost every case. For the soft voting strategy, the OFA² ensembles perform better in same regions and the OFA ensembles perform better in others.

Table 4.4 – Ensembles with individuals from OFA and OFA² searches, considering the sum of the latencies of all individuals on the ensemble.

	Latency (sum)	
	Hard voting	Soft voting
2 individuals		
3 individuals		
4 individuals		

Continued on next page

Table 4.4: Ensembles with individuals from OFA and OFA² searches, considering the sum of the latencies of all individuals on the ensemble (Continued).

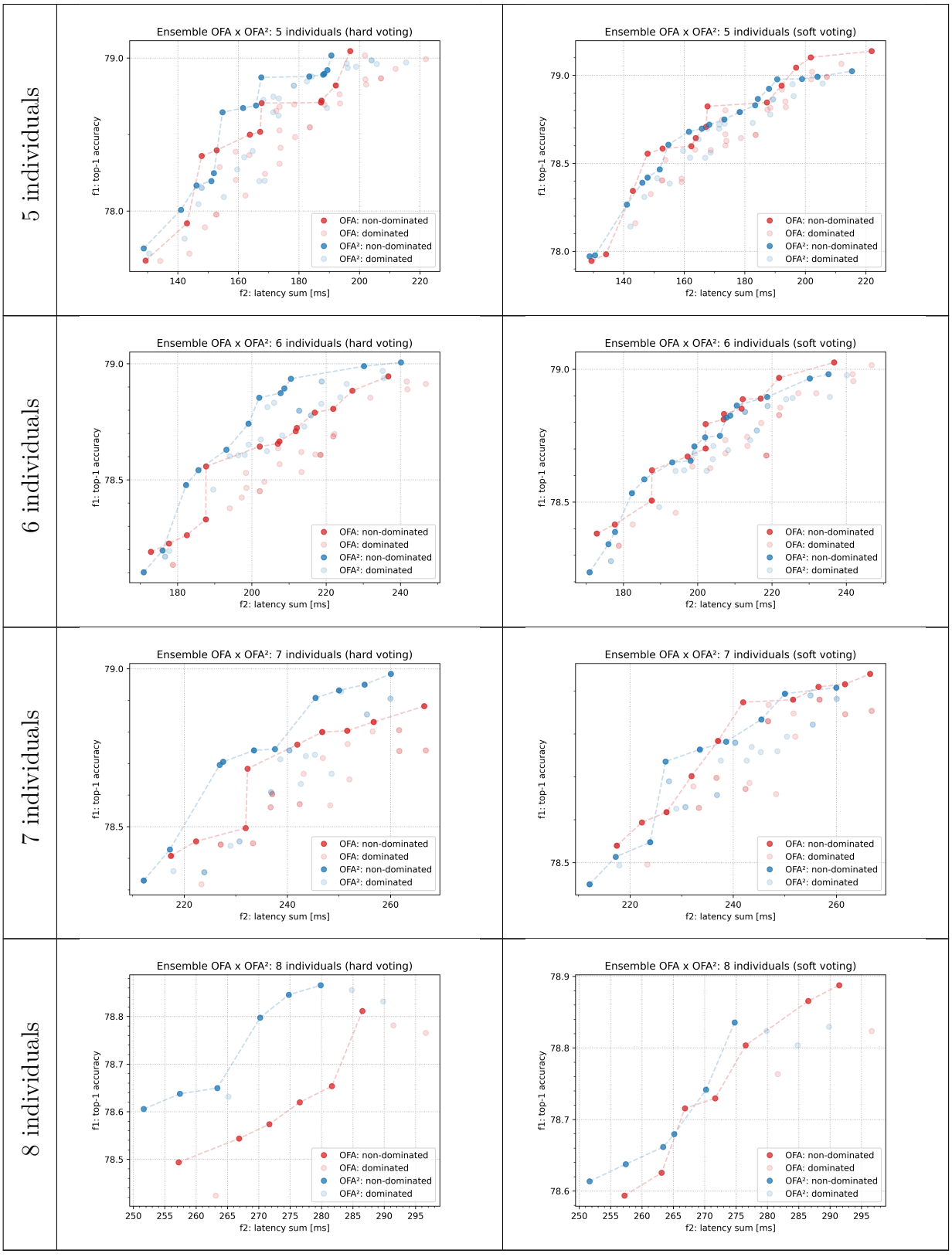
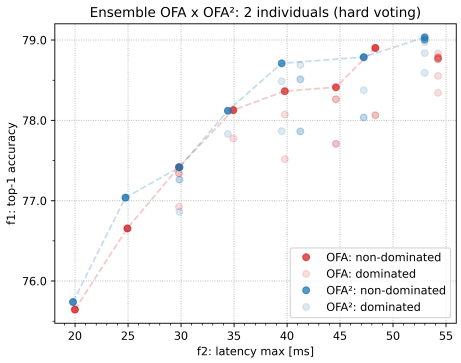
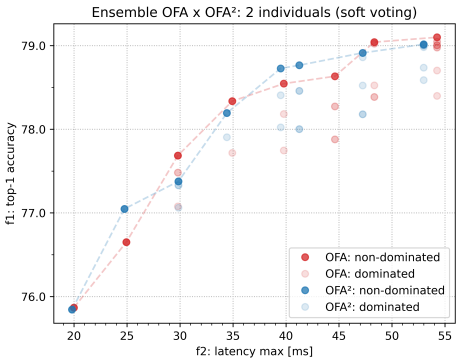
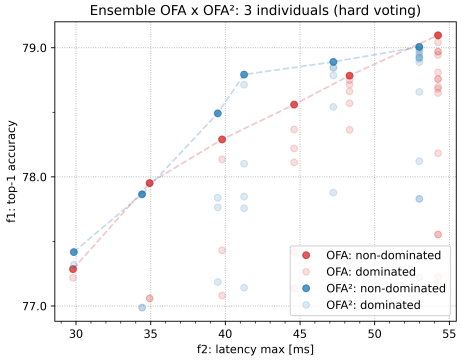
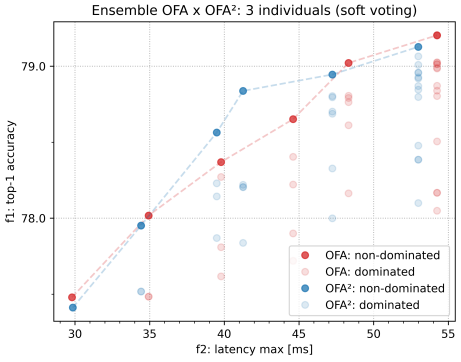
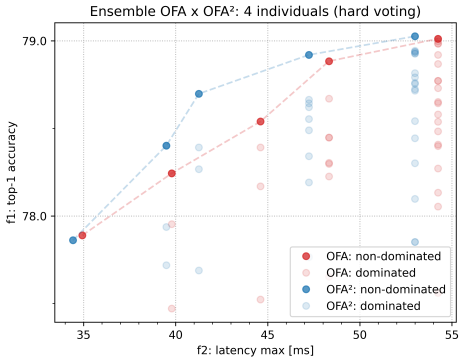
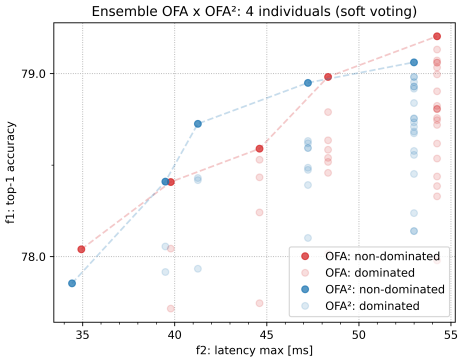
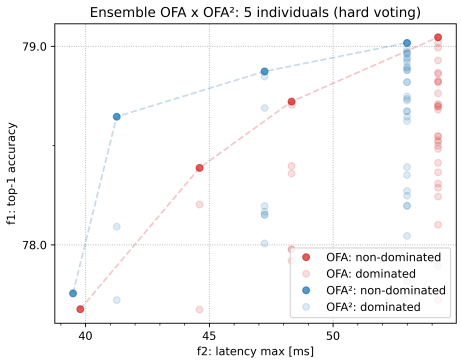
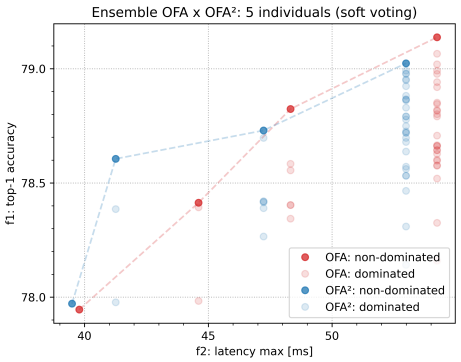
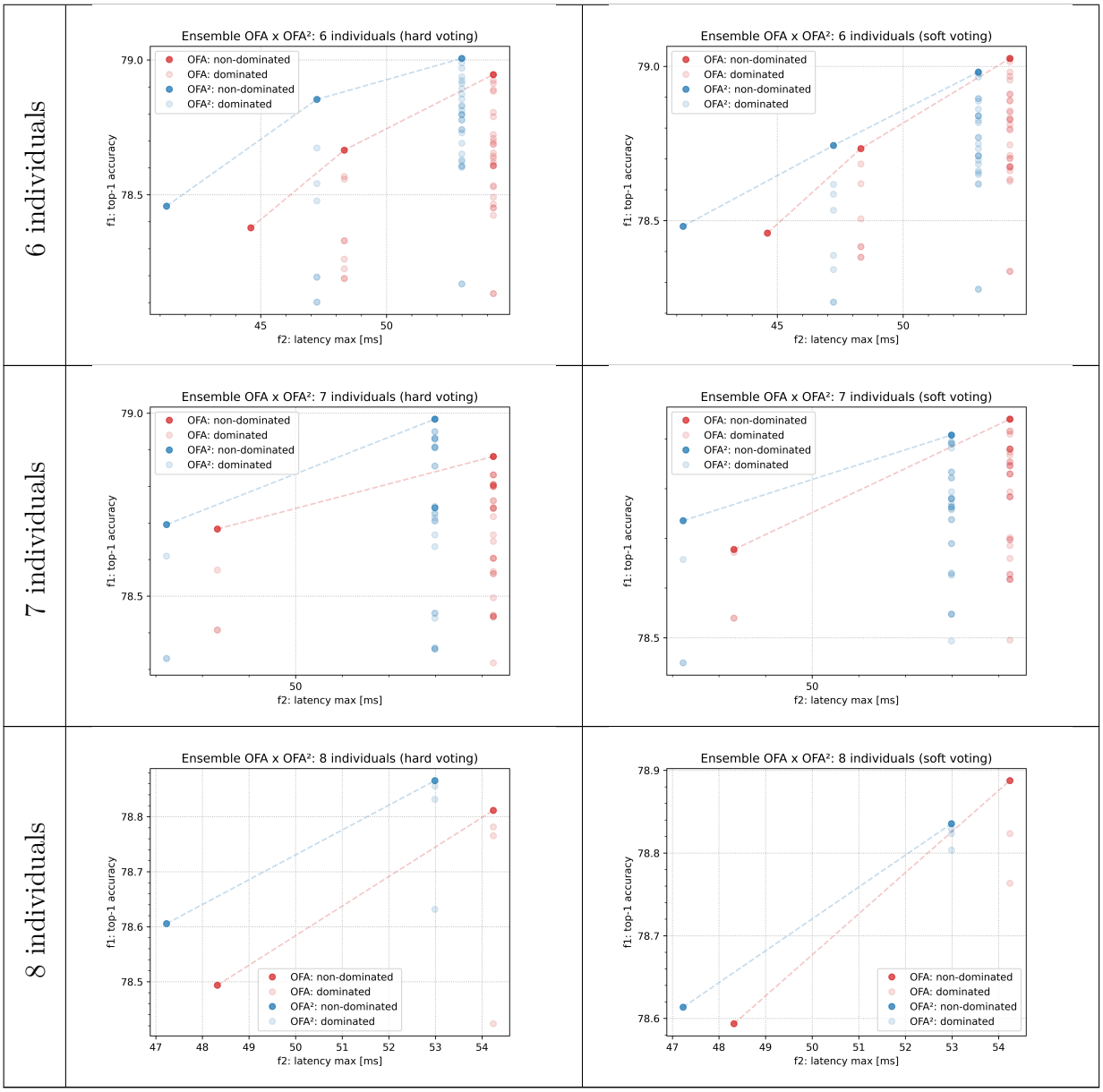


Table 4.5 – Ensembles with individuals from OFA and OFA² searches, considering the maximum latency of the individuals on the ensemble.

	Latency (max)	
	Hard voting	Soft voting
2 individuals		
3 individuals		
4 individuals		
5 individuals		

Continued on next page

Table 4.5: Ensembles with individuals from OFA and OFA² searches, considering the maximum latency of the individuals on the ensemble (Continued).



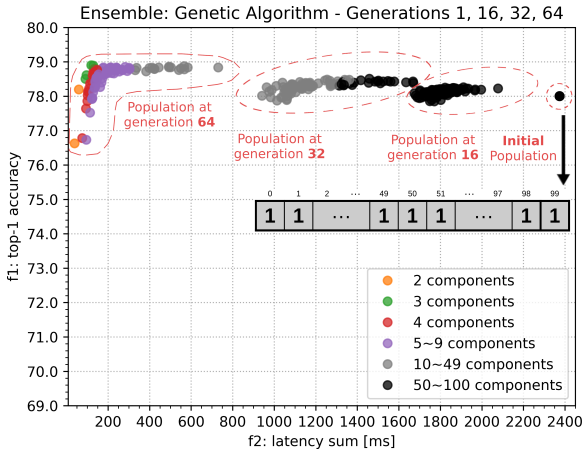
4.2.5 Selecting the components of the ensemble by evolutionary multi-objective approaches

For the next experiments, instead of choosing the individuals of the ensemble a priori, we formulate and solve the problem of choosing the participants of the ensemble as a multi-objective optimization problem. The population size is defined to be 100 individuals, where each individual represents an ensemble composed of one or more neural networks obtained from the OFA² search method. More details of the genetic algorithm used was described in section 3.3.4. We conceived experiments considering both the summed and maximum latency for the ensembles.

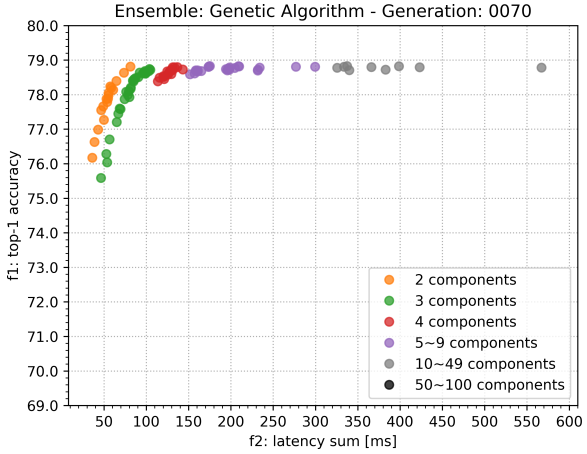
Summed latency

In this first part we define the latency of the ensemble to be equal to the sum of the latencies of all neural networks participating on the ensemble. The initial population consists of individuals with all genes with value 1, meaning that all individuals of the first population are ensembles with all neural networks available participating. The only restriction used during the optimization is that an individual must have at least two genes with value one, meaning that no single architectures are allowed. The single black point on the extreme right of Figure 4.27a illustrates the initial population. We have just a single point because all ensembles represented by this first population are equal, having therefore the same accuracy and latency. In the same figure, we can see the populations for generations 16, 32 and 64, represented by the different cloud of points, as indicated. These generations were chosen as power of two to illustrate the non-linear characteristic of the evolution. At generation 16, only ensembles with 50 or more components are present. At generation 32, most of the ensembles have between 10 and 49 components, and at generation 64, we start seeing ensembles with 2, 3 and 4 neural networks. We then plot the individuals at generations 70 and 128 in Figures 4.27b and 4.27c, respectively. Note that the latency scale has been changed to fit the populations more accurately. At generation 70, the majority of the ensembles have between 2 and 4 neural networks, and at generation 128, almost all the population are composed of 2 individuals. Finally, Figure 4.27d illustrates the final population.

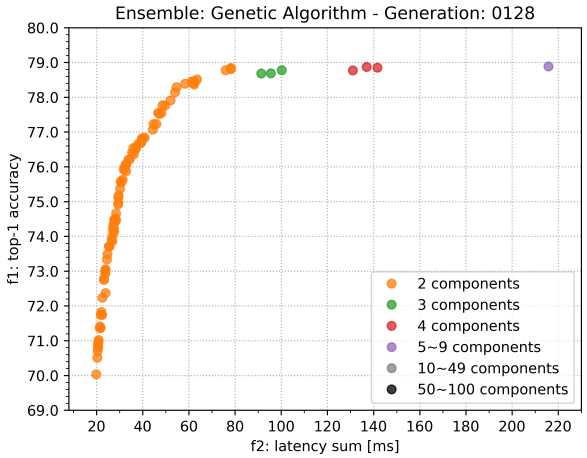
Figure 4.28 compares the final population of the evolutionary process against the single architectures obtained by the OFA^2 search, used as the foundation of the ensembles. We can see that the obtained ensembles form a typical Pareto-front for multi-objective optimization problems with two conflicting objectives. Most of these ensembles are, however, dominated by the single architectures, except for ensembles with more than 80 ms of total latency, where the accuracy of the ensembles surpass the accuracy of the single architectures. This dominance of single architectures can be explained due to the difference of scale between the objective functions. Take for example the two smallest neural networks candidates on participating of the ensembles. The first one presents a latency of 9.9 ms and accuracy of 69.84%, while the second one presents a latency of 10.0 ms and accuracy of 70.02%. When summing the latencies of these two architectures, we have a total latency of 20.0 ms. If we take the architecture with the highest latency under 20 ms, we have a single neural network with 75.94% of accuracy and 19.7 ms of latency. It is unlikely that and ensemble of two architectures with around 70% of accuracy surpass the 76% of the single architecture with equivalent latency, even more considering that these two neural networks of around 70% of accuracy are probably similar to each other on the decision space. We argue that the sum of latencies penalize too much the ensembles, so that it seems to be better the use of single architectures instead.



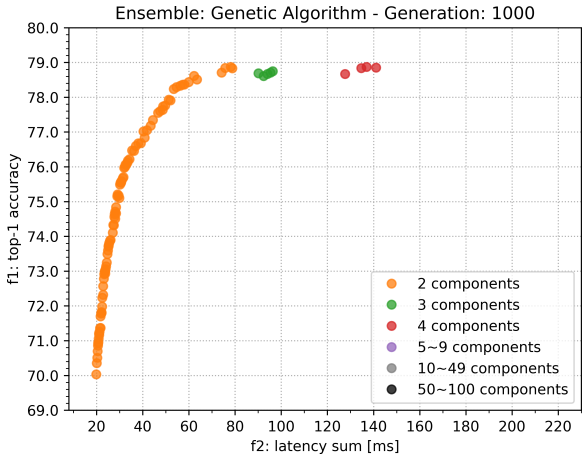
(a) Progression of the initial populations.



(b) Population at generation 70.



(c) Population at generation 128.



(d) Final population at generation 1,000.

Figure 4.27 – Progression of the NSGA-II populations of individuals reprensenting ensembles for the summed latencies approach. a) Generations 0, 16, 32 and 64. b) Generation 70. c) Generation 128. d) Last population at generation 1,000

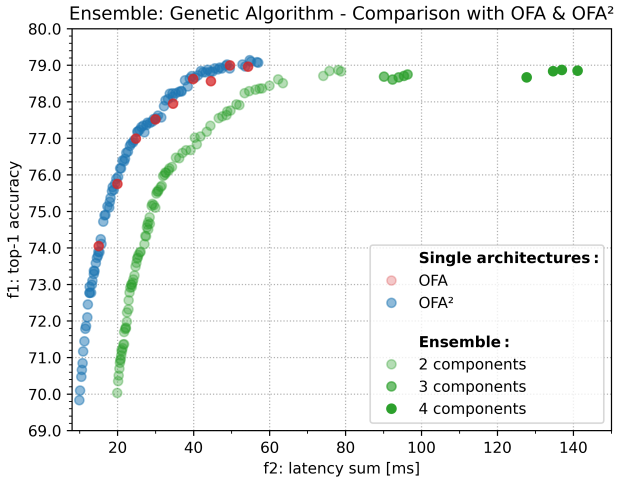
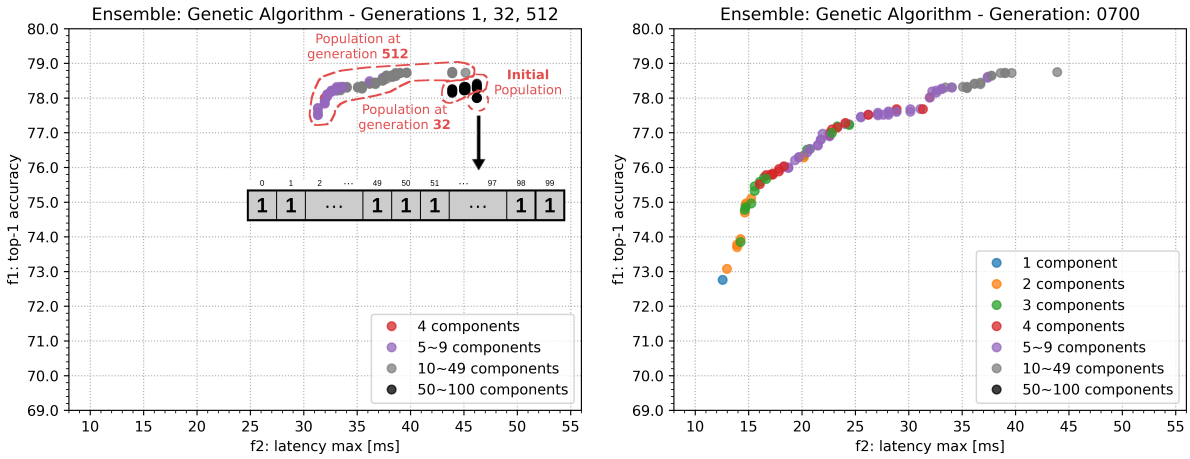


Figure 4.28 – Comparison between ensembles and single architectures (summed latency).

Maximum latency

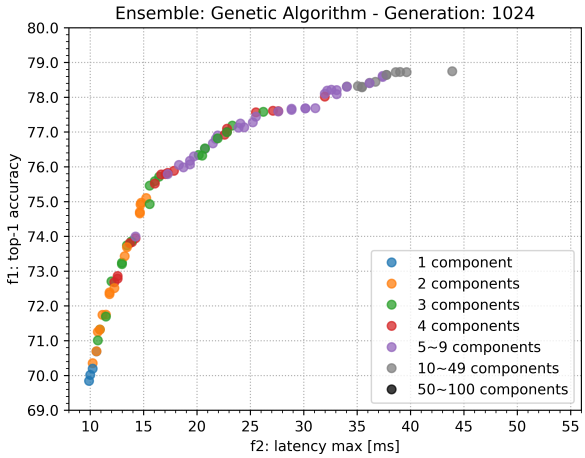
To alleviate the problem of difference in scale between latency and accuracy presented during the summed latency approach, we propose the same experiment but taking the maximum latency of the neural networks participating on the ensemble to be the latency of the ensemble itself. Notice that this hypothesis is motivated by the possibility of executing all the components of the ensemble simultaneously, in parallel. The lowest black point in Figure 4.29a illustrates the initial population, again with all genes equal to 1, meaning that all neural network candidates are composing the ensembles. Then, in the same figure we see two clouds of points illustrating the populations at generations 32 and 512, as indicated. At generation 32 we still have only ensembles with more than 50 components, while at generation 512 the ensembles present less than 50 neural networks. Figures 4.29b and 4.29c illustrates the population at generations 700 and 1,024, respectively. At generation 700 we start seeing ensembles with different number of components and at generation 1,024 some architectures within the range of 15 ms were already found. Figure 4.29d shows the last population of ensembles, after 2,000 generations. On the contrary to what was done with the summed latency approach, here we do not restrict the optimization to ensembles with 2 components or more. In fact, we can see that in the final population there are some individuals that are single architectures (in blue), meaning that other ensembles with more neural networks with lower latency perform actually worse than that specific single neural network.

Figure 4.30 compares the ensembles found by the evolutionary algorithm against the OFA^2 single architectures used as the foundation to form the ensembles. Here we can clearly see advantages of the ensembles over the single architectures, with the former dominating the latter for almost all latencies. The only exception happens at the beginning of the curve, where the accuracies and latencies of ensembles and single architectures are similar. In fact, some of the ensembles found by the evolutionary algorithm in this region are actually single architectures. This can be explained since the pool of neural networks to form the ensembles increases proportionally with the latency of the ensembles. For example, for the ensemble with the highest latency, all neural networks are available to join the ensemble, while at the ensemble with lowest latency, there is no ensemble at all, with only one neural network being available to form the "ensemble". It is interesting to note that the evolutionary algorithm tends to reduce the number of neural networks of the ensembles along generations, even though the first generation started with all neural networks being part of the ensembles. This indicates that ensembles with fewer components may perform better than ensembles with all neural networks (Bonab & Can, 2019).

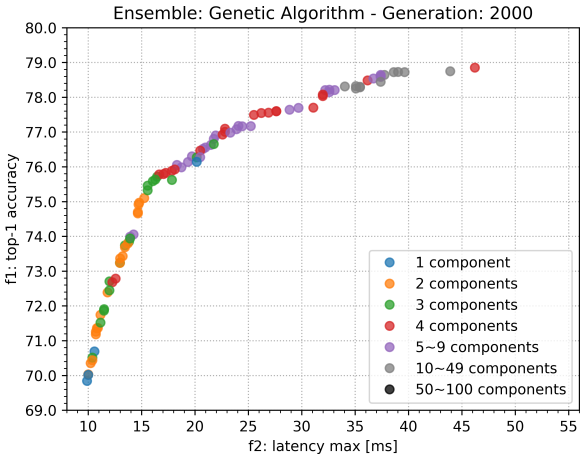


(a) Progression of the initial populations.

(b) Population at generation 700.



(c) Population at generation 1,024.



(d) Final population at generation 2,000.

Figure 4.29 – Progression of the NSGA-II populations of individuals reprensenting ensembles for the maximum of latencies approach. a) Generations 1, 32 and 512. b) Generation 700. c) Generation 1024. d) Last population at generation 2,000.

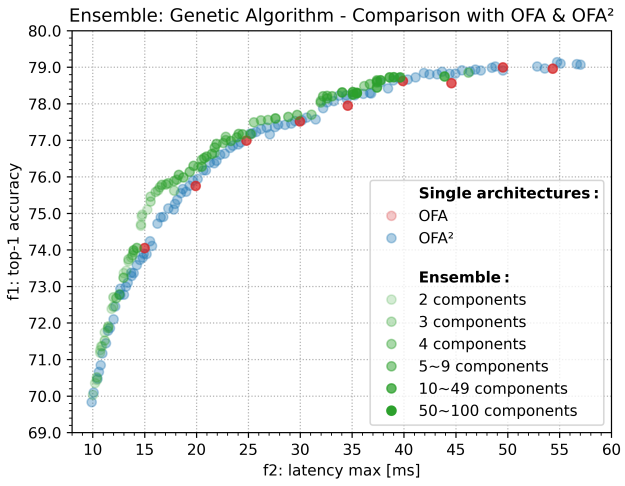


Figure 4.30 – Comparison between ensembles and single architectures (max latency).

Table 4.6 show the results for the OFA, OFA² and the genetic ensembles (represented by the abbreviation G.E.). We can see that for most of the latency constraints, the genetic ensembles considering the maximum latency performs better than OFA and OFA². On the other hand, for the highest latencies, the OFA and OFA² perform better than the ensembles. This could be explained by the fact that the evolutionary algorithm populates mid and lower latencies regions with more individuals than higher latencies, which could be alleviated by performing a local search in this region. Again, the computational costs of all methods are negligible when compared with the cost of training the Once-for-All super-network (1,200 GPU hours).

Table 4.6 – Comparison of ImageNet results between different hardware-aware NAS search methods.

	ImageNet Top-1% under latency constraints									Search cost
method	10 ms	15 ms	20 ms	25 ms	30 ms	35 ms	40 ms	45 ms	50 ms	GPU hour
OFA	N/A	74.05	75.75	76.99	77.52	77.95	78.62	78.62	79.00	0.83
OFA²	69.83	73.79	75.76	76.89	77.46	78.23	78.64	**78.88**	**79.02**	**0.01**
ensemble (sum)	N/A	N/A	70.03	73.59	75.21	76.22	76.68	76.34	77.76	0.98
ensemble (max)	**70.03**	**74.97**	**76.18**	**77.20**	**77.70**	**78.31**	**78.73**	78.75	78.75	1.83

5 Concluding Remarks

The starting principle of this work was provided by OFA (Cai et al., 2020), which has promoted the decoupling of training and search stages in NAS, thus making the search stage of negligible cost, when compared to the Once-for-All training of the super-network. In fact, any subnetwork that is sampled from the search space is already trained, thus making of low cost even a more elaborate search procedure. Therefore, there is room for the multi-objective search, which was accomplished with the proposed method OFA^2. This proposal not only finds better architectures in terms of top-1 accuracy and latency, but also returns a set of solutions instead of a single one, each of them being optimal considering a specific trade-off among the objective functions.

We then provide a series of experiments related to ensembles comparing the different search techniques that originated the neural network candidates. We show that ensembles with OFA^2 searched architectures consistently outperform the architectures from random search, and also outperform the ensembles from architectures found by the original OFA search for most of the cases, while being computationally much cheaper, since all networks from OFA^2 are searched at once in a single run.

Furthermore, besides the multi-objective search OFA^2, we also propose a multi-objective selection called OFA^3. The OFA^3 proposal involves a cascade of three Once-for-All mechanisms during the NAS: a single training step (provided by OFA), a single search step to populate an approximation of the Pareto frontier (provided by OFA^2), and a single selection step over the output of OFA^2 to compose the ensemble of efficient learning models. This is a remarkable achievement due to two main reasons:

I. The whole computational cost for the search stage remains of a reduced amount when compared to the Once-for-All training of the super-network, even performing a cascade of two consecutive multi-objective searches;

II. The multi-objective selection of efficient components (taken from the output of OFA^2) for the ensemble, which is the main contribution of OFA^3, is motivated by three main factors:

   a) The guaranteed presence of distinct trade-offs among the candidate components provided by OFA^2, given that they populate an approximation of the Pareto frontier;

   b) The assurance that they are independent models and can operate fully in parallel;

   c) The possibility of automatically choosing just a subset of the efficient models produced by OFA^2 as components of the best ensemble. Those are the main

motivation to support the gain in performance when compared with, for instance, a single model of the same size of the whole ensemble, which would not be implementable by resorting to independent fully parallelizable subnetworks.

This framework discovers architectures with improved top-1 accuracy and latency. All the source code has been made available, and we show in the experiments that OFA^3 compares favorably with the architectures found by the original OFA and OFA^2, in the sense of achieving higher accuracy for the same latency threshold, supposing that the components of the ensemble are run in parallel, given that they are independent models. Additionally, the evolutionary algorithm adopted by OFA^3 is able to determine the appropriate number of components of the ensemble, being a subset of the efficient models provided by OFA^2, while keeping constraints (such as latency) within specific bounds along the Pareto frontier.

5.1 Future Works

There are some directions in order to improve the work presented here even further. The most obvious next step is to fine-tune the architectures found by the OFA^2 search, with the goal of improving the neural networks performance. Generating latency predictors for different hardware, and then applying the multi-objective optimization to these hardwares other than the Samsung Galaxy Note 10 also seems a natural next step. In this work we used mainly the predicted latency as the second objective function of the multi-objective optimization alongside with the predicted accuracy. Working with FLOPS instead of latency is also an interesting idea. Finally, being able to train an Once-for-All super-network for other application domains other than computer vision, such as time series or natural language processing, sounds also promising.

6 References

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., & Le, Q. (2018). Understanding and Simplifying One-Shot Architecture Search. *Proceedings of the 35th International Conference on Machine Learning*, 550–559. <https://proceedings.mlr.press/v80/bender18a.html>

Bender, G., Liu, H., Chen, B., Chu, G., Cheng, S., Kindermans, P.-J., & Le, Q. V. (2020). *Can Weight Sharing Outperform Random Architecture Search? An Investigation With TuNAS*. 14323–14332. <https://openaccess.thecvf.com/content_CVPR_2020/html/Bender_Can_Weight_Sharing_Outperform_Random_Architecture_Search_An_Investigation_With_CVPR_2020_paper.html>

Beume, N., Naujoks, B., & Emmerich, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, *181*(3), 1653–1669. <https://doi.org/10.1016/j.ejor.2006.08.008>

Blank, J., & Deb, K. (2020). Pymoo: Multi-Objective Optimization in Python. *IEEE Access : Practical Innovations, Open Solutions*, *8*, 89497–89509. <https://doi.org/10.1109/ACCESS.2020.2990567>

Bonab, H., & Can, F. (2019). Less Is More: A Comprehensive Framework for the Number of Components of Ensemble Classifiers. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(9), 2735–2745. <https://doi.org/10.1109/TNNLS.2018.2886341>

Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2022, February 10). *SMASH: One-Shot Model Architecture Search through HyperNetworks*. International Conference on Learning Representations. <https://openreview.net/forum?id=rydeCEhs->

Brown, G. (2004). *Diversity in neural network ensembles*. <https://www.semanticscholar.org/paper/Diversity-in-neural-network-ensembles-Brown/b2329bfeaff2c9edbe4891ad56e4a4e03ad4fa59>

Brown, G., Wyatt, J., Harris, R., & Yao, X. (2005). Diversity creation methods: A survey and categorisation. *Information Fusion*, *6*(1), 5–20. <https://doi.org/10.1016/j.inffus.2004.04.004>

Burke, E. K., & Kendall, G. (2014). *Search Methodologies*. Springer US. <https://doi.org/10.1007/978-1-4614-6940-7>

Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2020). Once for All: Train One Network and Specialize it for Efficient Deployment. *8th International Conference on Learning Representations*. Eighth International Conference on Learning Representations. <https://iclr.cc/virtual_2020/poster_HylxE1HKwS.html>

Cai, H., Zhu, L., & Han, S. (2019). ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *7th International Conference on Learning Representations*.

International Conference on Learning Representations. <https://openreview.net/forum?id=HylVB3AqYm>

Chen, X., Xie, L., Wu, J., & Tian, Q. (2019). *Progressive DARTS: Bridging the Optimization Gap for NAS in the Wild* (Version 2). arXiv. <https://doi.org/10.48550/arXiv.1912.10952>

Deb, K. (2014). Multi-objective Optimization. In E. K. Burke & G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 403–449). Springer US. <https://doi.org/10.1007/978-1-4614-6940-7_15>

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*, 182–197. <https://doi.org/10.1109/4235.996017>

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. <https://doi.org/10.1109/CVPR.2009.5206848>

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural Architecture Search: A Survey. *Journal of Machine Learning Research*, *20*(55), 1–21. <http://jmlr.org/papers/v20/18-598.html>

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, *4*(1), 1–58. <https://doi.org/10.1162/neco.1992.4.1.1>

Green, S., Vineyard, C. M., Helinski, R., & Koç, Ç. K. (2019). *RAPDARTS: Resource-Aware Progressive Differentiable Architecture Search* (Version 1). arXiv. <https://doi.org/10.48550/arXiv.1911.05704>

Guerreiro, A. P., Fonseca, C. M., & Paquete, L. (2021). The Hypervolume Indicator: Computational Problems and Algorithms. *ACM Computing Surveys*, *54*(6), 119:1–119:42. <https://doi.org/10.1145/3453474>

Guo, Y., Chen, Y., Zheng, Y., Chen, Q., Zhao, P., Chen, J., Huang, J., & Tan, M. (2021). *Pareto-Frontier-aware Neural Architecture Generation for Diverse Budgets*. arXiv. <https://doi.org/10.48550/arXiv.2103.00219>

Hansen, L., & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(10), 993–1001. <https://doi.org/10.1109/34.58871>

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825, 7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

<https://doi.org/10.1109/CVPR.2016.90>

Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., & Le, Q. (2019). Searching for MobileNetV3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 1314–1324. <https://doi.org/10.1109/ICCV.2019.00140>

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-Excitation Networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7132–7141. <https://doi.org/10.1109/CVPR.2018.00745>

Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing. <https://doi.org/10.1007/978-3-030-05318-5>

Jiang, Y., Hu, C., Xiao, T., Zhang, C., & Zhu, J. (2019). Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3585–3590. <https://doi.org/10.18653/v1/D19-1367>

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. 60. <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, *25*. <https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>

Kuncheva, L. I., & Whitaker, C. J. (2003). Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning*, *51*(2), 181–207. <https://doi.org/10.1023/A:1022859003006>

Kuncheva, L., Whitaker, C., Shipp, C., & Duin, R. (2003). Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis & Applications*, *6*(1), 22–31. <https://doi.org/10.1007/s10044-002-0173-7>

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., & Li, Z. (2019). *DARTS+: Improved Differentiable Architecture Search with Early Stopping* (Version 2). arXiv. <https://doi.org/10.48550/arXiv.1909.06035>

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, *18*(185), 1–52. <http://jmlr.org/papers/v18/16-558.html>

Li, L., & Talwalkar, A. (2020). Random Search and Reproducibility for Neural Architecture Search. *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, 367–377. <https://proceedings.mlr.press/v115/li20c.html>

Liu, H., Simonyan, K., & Yang, Y. (2018). *DARTS: Differentiable Architecture Search* (Version 2). arXiv. <https://doi.org/10.48550/arXiv.1806.09055>

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, *4*(4), 380–387. <https://doi.org/10.1109/4235.887237>

Luo, R., Tian, F., Qin, T., Chen, E., & Liu, T.-Y. (2018). Neural Architecture Optimization. *Advances in Neural Information Processing Systems*, *31*. <https://papers.nips.cc/paper_files/paper/2018/hash/933670f1ac8ba969f32989c312faba75-Abstract.html>

Lu, Z., Deb, K., Goodman, E., Banzhaf, W., & Boddeti, V. N. (2020). NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search. *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, 35–51. <https://doi.org/10.1007/978-3-030-58452-8_3>

Lu, Z., Whalen, I., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W., & Boddeti, V. N. (2020). *NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm (Extended Abstract)*. *5*, 4750–4754. <https://doi.org/10.24963/ijcai.2020/659>

Lu, Z., Whalen, I., Dhebar, Y., Deb, K., Goodman, E. D., Banzhaf, W., & Boddeti, V. N. (2021). Multiobjective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification. *IEEE Transactions on Evolutionary Computation*, *25*(2), 277–291. <https://doi.org/10.1109/TEVC.2020.3024708>

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, *32*. <https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html>

Perrone, M. P., & Cooper, L. N. (1995). When networks disagree: Ensemble methods for hybrid neural networks. In *How We Learn; How We Remember: Toward an Understanding of Brain and Neural Systems: Vol. Volume 10* (pp. 342–358). WORLD SCIENTIFIC. <https://doi.org/10.1142/9789812795885_0025>

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the Aaai Conference on Artificial Intelligence*, *33*(01), 4780–4789. <https://doi.org/10.1609/aaai.v33i01.33014780>

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., & Kurakin, A. (2017). Large-Scale Evolution of Image Classifiers. *Proceedings of the 34th International Conference on Machine Learning*, 2902–2911. <https://proceedings.mlr.press/v70/real17a.html>

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. <https://doi.org/10.1038/323533a0>

Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv. <http://arxiv.org/abs/1409.1556>

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. <https://doi.org/10.1109/CVPR.2015.7298594>

White, C., Neiswanger, W., & Savani, Y. (2021). BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. *Proceedings of the Aaai Conference on Artificial Intelligence*, *35*(12, 12), 10293–10301. <https://doi.org/10.1609/aaai.v35i12.17233>

Wistuba, M., Rawat, A., & Pedapati, T. (2019). *A Survey on Neural Architecture Search*. arXiv. <https://doi.org/10.48550/arXiv.1905.01392>

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated Residual Transformations for Deep Neural Networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5987–5995. <https://doi.org/10.1109/CVPR.2017.634>

Xie, S., Zheng, H., Liu, C., & Lin, L. (2022, February 10). *SNAS: Stochastic neural architecture search*. International Conference on Learning Representations. <https://openreview.net/forum?id=rylqooRqK7>

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., & Xiong, H. (2020, April). *PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search*. Eighth International Conference on Learning Representations. <https://iclr.cc/virtual_2020/poster_BJlS634tPr.html>

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., & Hutter, F. (2019). NAS-Bench-101: Towards Reproducible Neural Architecture Search. *Proceedings of the 36th International Conference on Machine Learning*, 7105–7114. <https://proceedings.mlr.press/v97/ying19a.html>

Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014* (pp. 818–833). Springer International Publishing. <https://doi.org/10.1007/978-3-319-10590-1_53>

Zhou, Z.-H., Wu, J., & Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, *137*(1), 239–263. <https://doi.org/10.1016/S0004-3702(02)00190-X>

Zhu, H., Zhang, H., & Jin, Y. (2021). From federated learning to federated neural architecture search: A survey. *Complex & Intelligent Systems*, *7*(2), 639–657. <https://doi.org/10.1007/s40747-020-00247-z>

Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. In *Tik report* (Vol. 103) [Report]. ETH Zurich. <https://doi.org/10.3929/ethz-a-004284029>

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, *3*(4), 257–271. <https://doi.org/10.1109/4235.797969>

Zoph, B., & Le, Q. (2017). Neural Architecture Search with Reinforcement Learning. *5th International Conference on Learning Representations*. International Conference on Learning Representations. <https://openreview.net/forum?id=r1Ue8Hcxg>

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning Transferable Architectures for Scalable Image Recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8697–8710. <https://doi.org/10.1109/CVPR.2018.00907>